Online Analysis of Dynamic Streaming Data

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Eileen Kühn

aus Luckenwalde

Tag der mündlichen Prüfung:	04.07.2017
Erster Gutachter:	Prof. Dr. Achim Streit
Zweiter Gutachter:	Prof. Dr. Günter Quast

Erklärung zur selbständigen Anfertigung der Dissertationsschrift

Hiermit erkläre ich, dass ich die Dissertationsschrift mit dem Titel

Online Analysis of Dynamic Streaming Data

selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Regeln zur Sicherung guter wissenschaftlicher Praxis am Karlsruher Institut für Technologie (KIT) beachtet habe.

Ort, Datum Eileen Kühn

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Distanzmessung dynamischer, semistrukturierter Daten in kontinuierlichen Datenströmen um Analysen auf diesen Datenstrukturen bereits zur Laufzeit zu ermöglichen. Aktuelle Studien zeigen, dass die Bedeutung der effizienten Analyse von Datenströmen semistrukturierter und unstrukturierter Daten, die beispielsweise durch Sensoren und mobile Geräte generiert werden, in Zukunft stark zunehmen wird. Besonders das Volumen der generierten Daten und die Anzahl paralleler Datenströme machen es unerlässlich, diese möglichst direkt auf Basis individueller Datenströme zu verarbeiten und auszuwerten.

Dies wird jedoch erschwert durch die Dynamik der Daten selbst: Nicht ausschließlich der aktuelle Messwert, sondern dessen Veränderung über die Zeit definieren die Charakteristiken der Daten. Im Speziellen ist diese Arbeit fokussiert auf das nutzerbasierte Monitoring und die Analyse von Anwendungen im Batchsystem des GridKa Daten- und Rechenzentrums: Diese Anwendungen werden modelliert als dynamische Bäume mit Attributen, welche die Hierarchie von Prozessen und deren Eigenschaften abbilden. Dieser Anwendungsfall dient exemplarisch zur Analyse von dynamischen Bäumen mit großem Volumen, dynamischen Attributen und unterschiedlichen Strukturen.

Bei der Distanzberechnung zwischen statischen Bäumen handelt es sich um ein etabliertes Forschungsgebiet, welches bereits seit Jahrzehnten vorangetrieben wird. Besonders die exakte Distanzberechnung zwischen Bäumen ist weiterhin Gegenstand aktueller Forschung und wird ständig verbessert. Die quadratische Komplexität aktueller Ansätze ermöglicht jedoch keine effiziente Anwendung auf Datenströmen dynamischer Bäume. Forschungen im Bereich abschätzender Distanzberechnungen mit linearer Komplexität sind zumeist durch Anwendungsgebiete wie die Zusammenführung von XML-Dokumentenbeständen motiviert, und nutzen Randbedingungen der zugrundeliegenden Datenstrukturen aus.

Die Annahmen der Verfahren zur abschätzenden Distanzberechnung sind optimiert für statische Bäume. Dies betrifft im Besonderen die Behandlung von Attributen: In statischen Bäumen werden nur Einzelwerte betrachtet, wohingegen in dynamischen Bäumen Zeitreihen von Attributswerten zu betrachten sind. Zudem sind die Verfahren spezialisiert auf einzelne Anwendungsfälle und funktionieren daher nur mit Datensätzen, welche spezifische Randbedingungen erfüllen. Bestehende Verfahren für die Abschätzung von Distanzen zwischen statischen Bäumen sind daher nicht direkt anwendbar für den vorliegenden Sachverhalt dynamischer Bäume in Datenströmen.

Die vorliegende Arbeit wird deshalb von den folgenden Herausforderungen motiviert: Die grundlegende Fragestellung ist, welche Einschränkungen für bestehende Verfahren der Distanzberechnung statischer Bäume nötig sind, um die Anwendung der Verfahren für dynamische Bäume zu ermöglichen. Diese Arbeit führt daher eine Formalisierung zur Distanzberechnung für statische und dynamische Bäumen ein, welche implizit die Randbedingungen für Datenströme sicherstellt. Die Betrachtung der Dynamik von Attributen einzelner Baumknoten erfordert die Darstellung komplexer Zusammenhänge. Dies ist die Motivation für eine Erweiterung der Formalisierung zur spezifischen Betrachtung der Verteilung von Attributswerten und deren inkrementellem Vergleich. Ferner bietet

die Festlegung auf ein spezifisches Kriterium zur Bewertung dynamischer Baumstrukturen nur wenig Flexibilität. Als solches ist die Einführung einer effizienten Methode zur gleichzeitigen Bestimmung mehrerer Distanzen essentiell für dynamische Bäume mit mehreren unabhängigen strukturellen Eigenschaften. Um zudem die Anwendbarkeit und Effizienz der Distanzmessung im Rahmen von Echtzeitanalysen zu ermöglichen, wird die Distanzmessung einem dichte-basierten Clustering zugrunde gelegt, um eine Anwendung des Clustering, einer Klassifikation, aber auch der Anomalieerkennung zu demonstrieren.

Die Ergebnisse dieser Arbeit basieren auf einer theoretischen Analyse der eingeführten Formalisierung von Distanzmessungen für dynamische Bäume. Diese Analysen werden unterlegt mit empirischen Messungen auf Basis von Monitoring-Daten von Anwendungen aus dem Batchsystem des GridKa. Hierzu wird mit Hilfe einer Referenzimplementierung die Skalierbarkeit der einzelnen Komponenten demonstriert und die Qualität der Ergebnisse mit Hilfe eines Referenzdatensatzes bewertet.

Die Evaluation des vorgeschlagenen Formalismus sowie der darauf aufbauenden Echtzeitanalysemethoden zeigen die Effizienz und Skalierbarkeit des Verfahrens. Zudem wird gezeigt, dass die Betrachtung von Attributen und Attributs-Statistiken von besonderer Bedeutung für die Qualität der Ergebnisse von Analysen dynamischer semistrukturierter Daten ist. Außerdem zeigt die Kombination von Methoden zur Distanzmessung aus der Literatur und eigener Ansätze, dass die Qualität der Ergebnisse durch eine unabhängige Kombination mehrerer Distanzen weiter verbessert werden kann. Insbesondere wird dadurch die Analyse sich über die Zeit ändernder Daten ermöglicht.

Abstract

The main topic of this thesis is the distance measurement of dynamic, semi-structured data in streaming environments. The goal is to enable the analysis of dynamic trees during the lifetime of the objects they describe. Recent studies show the increasing importance of efficient analyses of data streams defining structured and unstructured data, which are for example generated by sensors or mobile devices. In specific, due to the volume of such data and the number of concurrent data streams, it is essential to analyse data from individual streams directly.

Such an analysis is complicated by the dynamic nature of data: Not only the current state but also its change over time defines the data as a whole. As an exemplary use case, this thesis considers a user-centric monitoring and analysis of applications deployed in the batch system of the GridKa data and computing centre: Such applications can be modelled as attributed dynamic trees, corresponding to the process hierarchy and their features. This use case offers the challenge of analysing large-scale dynamic trees, frequent attribute changes, and multiple structural characteristics.

Distance measurement of static trees is a well-established field of research. Static distance measures have been researched for decades. Especially precise distance measures between trees is a focus of current research and progressively refined. The quadratic time and space complexity make the application on data streams and dynamic trees unfeasible. Research of approximating distance measures enables linear complexity. Such approximations are usually focused on specific use cases such as integration of XML documents, exploiting the constraints of the underlying data structures.

Assumptions used by approximating distance measures are optimised for static trees. This limits the description of attributes: Static trees can only express individual attribute values, whereas dynamic trees correspond to time series of values. Additionally, approximating approaches specialise on specific use cases. As such, they are only applicable for data meeting specific constraints. Thus, existing approaches are not directly applicable to the use case of dynamic, attributed trees in data streams.

As a result, this thesis addresses the following three main challenges: Fundamentally, a set of minimal constraints required to apply existing distance measures onto dynamic trees must be derived. To address this, this thesis introduces a formalisation for approximating distance measures on static and dynamic trees. This formalisation implicitly guarantees constraints for analyses of dynamic trees and streams of tree data. Furthermore, the change of attributes of individual tree vertices requires a representation of changing values. Changing attributes are addressed by an extension of the formalisation, representing vertex attributes as distributions of values, which can be compared incrementally. Additionally, using a single criterion to rate dynamic structural features offers only a little flexibility. This motivates an efficient method to derive multiple distances concurrently. In turn, this allows reflecting multiple independent characteristics of dynamic trees. Finally, application of distance measures must allow deriving an appropriate classification and outlier detection for the entities represented by trees. Classification and outlier detection are enabled by extending a density-based clustering to handle dynamic tree distances as well as deriving

an approximate but monotonic distance measure allowing for a classification at runtime.

The results presented in this thesis are founded on a theoretical analysis of features of the introduced formalism for distance measurements of dynamic trees. This analysis is supported by empirical studies using monitoring data collected on applications deployed in the GridKa batch system. A reference implementation is used to demonstrate the accuracy and scalability of the approach itself. The feasibility and quality of the approach are derived from a comparison against manually composed reference data.

The evaluation of the formalisation and implementation show the scalability, efficiency, and accuracy of the approach. Also, the analysis of attribute-based extensions shows the improvement for identifying and distinguishing classes of semi-structured data. Furthermore, the general applicability and robustness of using multiple independent distances measures are demonstrated. Overall, the evaluation shows that the approach is scalable for stream analyses, allows to distinguish classes by attributes and structural features, and enables an analysis at runtime.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Increasing access to and dependence on streaming data drives the growing interest in computationally efficient and scalable analysis tools and algorithms. A major difference of streaming data compared to static data is the dynamic characteristics. As such, incremental algorithms and data structures are required to enable online analysis on dynamic streaming data. In this thesis, we propose and demonstrate an incremental approach to compute distances on dynamic trees in a streaming environment.

The context of this thesis is the monitoring of network traffic of High Energy Physics batch jobs in the GridKa data and computing centre at Steinbuch Centre for Computing. Each batch job is composed of a hierarchy of multiple processes: starting and finishing processes cause the hierarchy of the batch job to grow and shrink over time. Some of these processes are providers for network traffic, which can be modelled as a time series of network traffic rates. Therefore, each batch job can be modelled as a dynamic tree with network traffic statistics represented by dynamic attributes.

With this approach of modelling batch jobs as dynamic trees, the monitoring of batch jobs at GridKa data and computing centre with its roughly 20 000 batch job slots produces thousands of concurrent streams of dynamic tree data. Based on this use case, this thesis aims for an online clustering and an outlier detection for batch jobs. The application of this work is the detection and limitation of negative impacts on the whole GridKa system itself from misbehaving batch jobs.

Both online clustering and outlier detection require an efficient similarity measure on dynamic trees. The analysis of differences and similarities for static tree-structured data has been an ongoing research topic for decades. Especially comparing two given trees to determine their similarity is important in a variety of contexts, including language processing, document similarity, or quantifying neuronal morphology. A common approach to measuring tree similarity is to determine the minimal cost of edit operations, that is vertex insertion, deletion, or renaming, to transform one tree into the other. This approach is NP-complete for unordered trees and computable in polynomial time for ordered trees. However, exact algorithms of this Tree Edit Distance measurements are not scalable to modern problem sizes.

To enable similarity measurements for static trees at a scale of thousands to millions of nodes, a range of approximation algorithms have been proposed in the past. The computational complexity of these algorithms ranges down to near-linear runtime. Such algorithms usually summarise a tree or apply tree sketching to derive a similarity measure, i.e. use a limited amount of space while still retaining relevant properties of a tree. While reducing complexity, most of these algorithms still assume static properties of trees. This restricts their applicability for modern problems: By design, most of these algorithms are not suitable for dynamic trees. As a result, they cannot be used for real-time data analysis, that is when streaming dynamic tree data.

Based on these challenges, this thesis is dedicated to the online analysis of dynamic trees in streaming scenarios. Moreover, the proposed methods are aimed at being scalable and efficient to tackle thousands of concurrent streams. We propose a combination of multiple

approaches including a composite identity profile and a similarity function for dynamic trees as well as sets of dynamic trees. To ensure the feasibility for streaming environments, we propose and utilise a formal framework in which similarity and distances are implicitly suitable for incremental stream processing. Thus, for each of our proposed similarity and distance measures, an incremental version is available to enable online processing. Results are evaluated on monitoring data of HEP batch jobs that were recorded at GridKa. However, the approach applies to any static or dynamic tree-structured data.

1.1. Main Contributions

This thesis necessarily covers a broad range of topics to enable online clustering and outlier detection for dynamic trees. However, the central question is how to efficiently measure and work with similarities and distances between attributed dynamic trees in streaming environments. While each contribution is applicable on its own, the assessment and demonstration of our technique require a complete monitoring, measurement, and analysis workflow. As such, this thesis addresses the following research sub-topics:

1.1.1. Requirements to identify workflows in an overlay batch system

Collaborations using the WLCG for data analysis have widely adopted the so-called pilot paradigm: placeholder jobs, the pilots, acquire processing resources to provide them to analysis applications. This creates an overlay batch system, isolating jobs from classical monitoring as a side effect.

Extracting information on jobs for sites requires an identification of jobs inside pilots. This necessitates a model of workflows and jobs and their relation to pilots. From this, requirements in monitoring data suitable to identify jobs can be derived.

1.1.2. Formalisation of distance measures for streaming dynamic trees

The hierarchical structure of pilots lends itself to the monitoring of pilots via their process trees. This necessitates an analysis of dynamic tree data. Ideally, tree distances are employed to allow comparing jobs for similarities and differences. Traditional tree distances for static trees are not suitable for the complexity, dynamics, and detail of our collected data.

A class of approximate tree distances based on decomposition offers appropriate complexity and detail. However, existing approaches rely on strict assumptions not met by dynamic trees, in which topology and features may vary over time. As such, a generalisation of decomposition methods suitable for dynamic trees must be created. This decomposition method must provide a basis to formulate specific distances to detect similarities and differences in our highly complex data.

1.1.3. Integration of attribute data for continuous vertex distances

A primary distinction of jobs in High Energy Physics (HEP) is network traffic of processes, separating data processing from simulation. With the view of jobs as dynamic trees, traffic can be expressed as attributes on each vertex. Such attributes are more complex than just the presence of vertices, extending the simple consideration of cardinality to continuous,

dynamic values. To the best our knowledge, there has been no dedicated research to express such attributes in dynamic trees.

Scalability requires an integration of vertex attributes into our framework of tree decomposition. This necessitates a simplification of values while preserving separability. Furthermore, the projection of vertex existence to a binary range must be generalised to attribute similarities in a continuous range.

1.1.4. Combination of distinct measures in streaming environments

As our data is collected in a real use case scenario, background noise is to be expected. Especially with highly structured data, not all observed relations originate from the logical features of the underlying jobs. Using complex relations to separate distinct features may not allow the suppression of noise.

Instead of a single multivariate distance measure, individual univariate measures have more flexibility to avoid noise. As such, an approach to combine multiple distinct measures in parallel is desirable. However, operation in a streaming environment requires finding constraints to preserve the required complexity of the approach.

1.1.5. Classification of semi-structured data in real-time

The practical application of our monitoring is the identification of groups of similar jobs. Such an identification has the goal of exposing information on jobs and identifying outliers for anomaly detection. To efficiently integrate with distributed, stream-based monitoring, the results of measurements should be sufficient.

The availability of distance measures lends itself to using distance-based clustering. However, clustering must be able to deal with dynamic data due to the online analysis. As there is a high number of concurrent distance measures, exploiting the incremental measurements to exclude clusters early is desirable.

1.2. Structure of this Thesis

The remainder of this thesis is organised as follows:

In Chapter 2 on page 7, we establish the use case providing the motivation for our work, namely the continuous monitoring of High Energy Physics batch jobs. First, we introduce the field of HEP and the characteristics of computing in the Worldwide LHC Computing Grid. Second, we discuss the motivation for monitoring of HEP batch jobs, complementing existing monitoring solutions. In the following, we introduce the GridKa data and computing centre, one of the biggest data and computing centres in the WLCG, where we have deployed a monitoring sensor for our research. Finally, we motivate the derivation of batch job behaviour by monitoring HEP batch jobs and summarise the challenges that arise in the context of monitoring in a production batch system.

Chapter 3 on page 23 is dedicated to the user-centric monitoring of jobs in a HEP batch system. We start by evaluating existing monitoring solutions for their applicability to requirements from HEP batch jobs. Following that, we develop a model of pilots, payloads, jobs and workflows, defining the information required to separate jobs via monitoring. This, in turn, is the basis for a custom monitoring tool, which defines the type and granularity of monitoring data – namely, attributed dynamic trees in streaming environments. Finally,

we explore the implications and constraints of this monitoring data on following analyses. Some of the materials presented in this Chapter were first formulated in

- Eileen Kuehn et al. "Monitoring Data Streams at Process Level in Scientific Big Data Batch Clusters." In: *BDC* (2014), pp. 90–95,
- Eileen Kuehn et al. "Analyzing data flows of WLCG jobs at batch job level". In: *Journal of Physics: Conference Series* 608.1 (May 2015), p. 012017, and
- Eileen Kuehn et al. "Active Job Monitoring in Pilots". In: *Journal of Physics: Conference Series* 664.5 (Dec. 2015), p. 052019.

The analysis of dynamic tree data streams from user-centric monitoring requires an appropriate distance measure to enable the clustering of jobs. In Chapter 4 on page 39 we, therefore, introduce available approaches to measure distances between trees. This analysis identifies tree decomposition methods as the most appropriate field of research to derive efficient distance measures for dynamic trees. In the following, we introduce a decomposition-based approach that builds on the concept of vertex identities. We focus on scalability and incremental approaches to enable processing for both static as well as dynamic trees. We conclude the Chapter with incremental distance measures building on tree decomposition. Some of the ideas and distance measures for dynamic trees were published in

- Eileen Kuehn and Achim Streit. "Online Distance Measurement for Tree Data Event Streams." In: *DASC/PiCom/DataCom/CyberSciTech* (2016), pp. 681–688.

In the following two Chapters we extend the proposed framework of distance measures for dynamic trees. In Chapter 5 on page 77, we extend the model of a dynamic tree for attributes and discuss implications to encoding and distance calculation. Two strategies to encode attributes for distance calculation are introduced and compared.

The second extension to the proposed framework is discussed in Chapter 6 on page 101. We examine the abstraction of tree encoding to ensemble-based methods. Furthermore, we illustrate the possibilities of ensembles by introducing two ensemble encoding strategies targeting the handling of noise as well as the robustness of distance measures.

In Chapter 7 on page 113, the application of our distance framework for an online analysis of our monitoring data is discussed. The concept of an online analysis to support a clustering to enable an outlier detection for dynamic trees was originally published in

- Eileen Kuehn. "Clustering Evolving Batch System Jobs for Online Anomaly Detection". In: *ICDMW '15: Proceedings of the 2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE Computer Society, Nov. 2015, pp. 1534–1535.

We further detail this concept by considering incremental distance measurements of many concurrent streams of dynamic trees for clustering and classification of data. In specific, we highlight how our approach allows the adoption of density-based clustering while staying suitable for online analysis. Building on this, we present a robust approximation of our incremental dynamic tree distance that allows classifying dynamic trees as outliers after only a fraction of events. Some of the ideas in this Chapter were first published in

- E Kuehn et al. "A scalable architecture for online anomaly detection of WLCG batch jobs". In: *Journal of Physics: Conference Series* 762.1 (Nov. 2016), p. 012002.

Finally, we evaluate both our approach itself and the means for online analysis in Chapter 8 on page 129. First, the characteristics of our approach are analysed with controlled conditions. We demonstrate that our approach is a suitable approximation of tree edit distances while providing scalability suitable for stream and online analysis. Furthermore, we show that additional features such as attributes can easily be covered. Finally, an exemplary analysis of HEP workflows shows the applicability of our work for realistic use cases. The premise of being able to identify jobs based on attributed dynamic trees representing process is validated. Additionally, the benefits of our approach, namely the effective reduction of complexity and early outlier detection, are shown.

After discussing and evaluating the proposed approach to the online analysis of dynamic tree distances, we arrive at final conclusions. In Chapter 9 on page 149, we summarise this thesis, confront its findings with the stated research objectives and draw lines of future research.

2. Background

Computing in HEP, especially the unprecedented amount of scientific data requiring global analysis efforts pose major challenges for providers of processing, network, and storage resources. Some of these challenges are rather unique to the considered use case of HEP while others are applicable to big data management and analysis in general. In the following, an overview of relevant requirements and general conditions regarding computing in HEP is given. Afterwards, the GridKa data and computing centre as one important resource provider for HEP is introduced. In particular, the current state of handling challenges posed by HEP is discussed. The Chapter concludes with a summary of challenges imposed by highly complex systems that builds the basis for the framework of this thesis.

2.1. Computing in High Energy Physics

HEP (also known as Particle Physics) is a branch of physics investigating the smallest particles and the fundamental forces governing their behaviour. Notably, it operates at both the energy and intensity frontier of physics research: High energy experiments allow for the discovery of new particles and phenomena, while high intensity experiments provide insights even for rare processes and particles. As a result, HEP research combines both high precision and high volume of data.

The field of HEP is largely driven by different particle collider experiments. Some prominent facilities are the Large Hadron Collider (LHC) at CERN [54], the upcoming SuperKEKB at the High Energy Accelerator Research Organization (KEK) [108] as well as possible future colliders [1, 52, 95, 219]. The most prominent, recent advancement of the field was the discovery of the Higgs boson at LHC [63, 64].

Research in HEP is based on collaborated efforts of multiple experiments and communities. These worldwide collaboration efforts rely on unique computing resources to enable their research: First, modern particle detectors provide data at enormous rates, producing data volumes unprecedented in research. Second, the unique challenges of storing and processing this data require custom computing infrastructures and software. Third, the collaborative analysis of this data requires an equally distributed infrastructure for thousands of scientists worldwide.

2.1.1. Data Flows

The experiments at the LHC accelerator generate enormous amounts of data. The data rates of modern particle detectors are considerably higher than the rate at which one can write data to mass storage. At the LHC, the rate is dictated by proton beams crossing each other at a frequency of 40 MHz. Each crossing results in a physics event recorded by the detector, with an approximate raw data size of 1 MB up to 1.5 MB. For all four major experiments the data flow is estimated to be about 25 GB/s. These figures massively exceed the maximum feasible rate for data acquisition as well as storage and require a data reduction by a factor of 10 000. [61, 218]

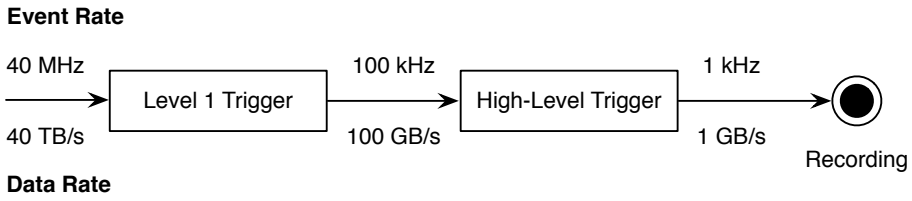


Figure 2.1.: Overview of the resulting event and data rates of the two-level trigger for the CMS experiment. The Level 1 hardware trigger reduces the event rate from the detector of approximately 40 MHz to 100 kHz. A software trigger, the so-called High-Level Trigger produces a final acceptance rate of 1 kHz before the data is recorded at CERN with a data rate of approximately 1 GB/s.

Each of the four experiments thus applies a real-time filtering and selection of events with the so-called *triggers*. Triggers inspect selective information from detector subsystems, evaluating whether events are suitable for offline physics analysis and should be recorded. For example the Compact Muon Solenoid (CMS) experiment uses a two-level approach: a trigger implemented in custom-build hardware, the so-called Level 1 trigger and a software trigger running in a dedicated processing farm, the High-Level Trigger (HLT). Figure 2.1 visualises this two-level approach and shows the data flows between the Level 1 trigger and HLT. The Level 1 trigger uses custom electronic systems to perform event selection based on events from raw hardware buffers of the detector and reduces the data rate by a factor of 1000 to less than 100 kHz. The HLT runs on a computing farm. It performs partial physics event reconstruction and achieves an output rate of 1 kHz [175]. The remaining events are recorded to servers at CERN with a data rate of approximately 1050 MB/s [53].

2.1.2. WLCG and Tiers

Analysing this volume of data requires massive amounts of computing resources. These include processing, storage, and network resources as well as human resources for operation and support. Providing all required resources at one site is unfeasible for a variety of reasons, from financial to organisational. Therefore, a global collaboration, the WLCG [202], was formed to set up and operate a geographically distributed infrastructure for simulation, processing, and analysis of the data of the LHC experiments.

As initially proposed by the Models of Networked Analysis at Regional Centres for LHC Experiments (MONARC) project [42], the WLCG defines a model for a hierarchical organisation of resource providers. The model classifies the different resource providers into so-called Tiers. Formally, three Tiers are defined in the hierarchy: a) Tier 0, b) Tier 1, and c) Tier 2. Figure 2.2 on the next page shows the different Tiers and its underlying hierarchy. This hierarchy specifies how data is distributed between the different sites. Consequently, data flows were defined to strictly follow the Tier hierarchy: The data from the LHC detectors enters the WLCG via the Tier 0. From there, the data is replicated to Tier 1 sites. In turn, Tier 1 sites provide data to Tier 2 sites as required.

As of 2017, the WLCG is composed of more than 170 computing centres in 42 countries [171]. Each of these sites provides resources, namely CPU, network, and storage, to the LHC collaborations. Resource provisioning with this number of independent sites relies on high-speed network infrastructure [42, 43]. Hence, the WLCG links sites in national and international grid infrastructures. These resources are highly heterogeneous. To ensure the

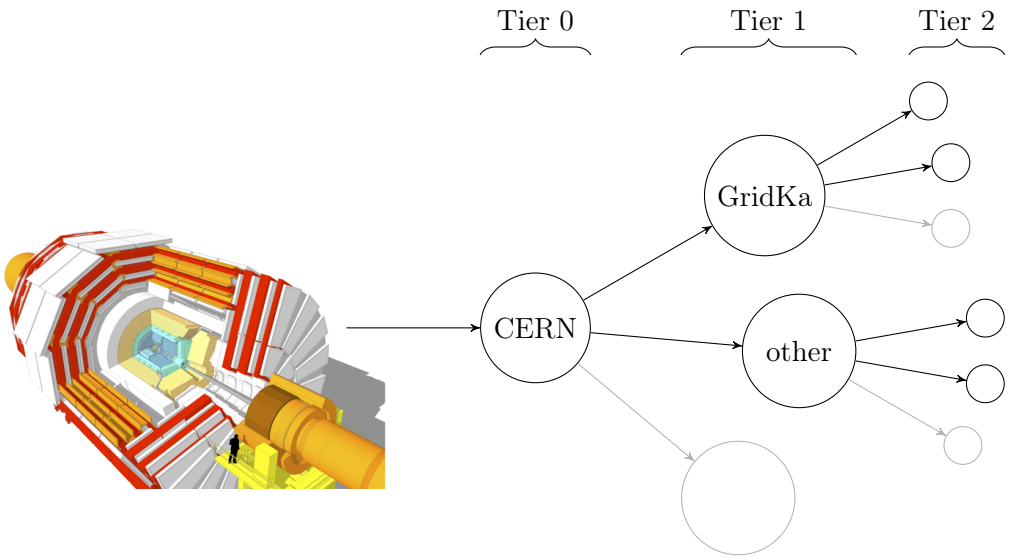


Figure 2.2.: Visualisation of WLCG Tier structure [227]. The Tiers are organised hierarchically, reflecting the tasks and quality measures assigned to each specific Tier. The data from the LHC detectors, such as the CMS detector [189], is provided by Tier 0 sites to Tier 1 sites via dedicated 100 Gbit/s links. Tier 1 sites keep parts of the data and following serve the different Tier 2 sites via national research and education networks. The computing centres that are assigned to a specific Tier need to fulfil the given requirements. In fact, there are more than 170 computing centres that compose the WLCG.

interoperability and provide access to these resources, a variety of middleware technologies and protocols is used [82, 87, 157].

The performance of network connections between sites has proven to allow for a more flexible data access than the initial, strictly hierarchical design. Following this, data access and processing has been adapted to utilise all available resources rather than adhering to the strict roles of the MONARC model. In particular, all LHC collaborations have adopted remote data access between sites when necessary [43]. Thus, the hierarchical data flows are no longer the defining feature for Tier affiliation. Instead, Tier 1 sites are defined by their long-term data archival and guarantees for quality of service.

2.1.3. Computing Model

Each of the LHC collaborations implements a *computing model*, defining the goals and policies to use distributed computing and storage resources of the WLCG [42, 43]. Sites provide storage and processing resources only at an abstract level: Computing elements (CEs) encapsulate batch processing farms, offering several thousand processing cores. Storage elements (SEs) represent file server pools with various underlying storage technologies, providing petabytes of storage. Each collaboration uses these elements differently to implement its computing model.

The computing models define how resources in the WLCG are aggregated, partitioned, and utilised for data storage and processing. While policies, protocols, and technologies differ between collaborations, the general approaches and goals are comparable. For this work,

an abstract model that describes the computing models of the major LHC collaborations is used. Since the use case is on monitoring HEP batch jobs, the abstract model focuses on *workflows* involving batch processing.

High Energy Physics Workflows

Analysis efforts of HEP collaborations require the processing of large datasets, each consisting of millions of physics events (compare Section 2.1.2 on page 8). Therefore, analysis efforts are divided into *workflows* which gradually process data in the WLCG. In turn, the workflows rely on trivial parallelisation, exploiting the statistical independence of each physics event. Thus, workflows usually consist of several thousand *jobs*, each working on a separate subset of data. Each job of a workflow is the same regarding algorithms and implementation, differing only in its values but not in its type of input.

The WLCG allows the concurrent, massively parallel processing of many workflows at once. Collaborations continually submit new workflows over long time frames to contribute to their overall processing and analysis effort. This means that sites are steadily processing many jobs of different workflows. Each job of a workflow is processed within a bounded time frame, for example 2.66 h on average for CMS. However, this implies a considerably larger amount of total computing time for the workflow as a whole due to parallelism in data processing [33]. The efficient processing of workflows thus requires high throughput of many jobs at once.

This processing model poses considerable technological and organisational challenges for collaborations. First, thousands of jobs and workflows of differing analyses must be coordinated and managed over a long time frame in the heterogeneous setup of the WLCG. Second, monitoring of failure and provenance are paramount to ensure the correct execution of all workflows and their repeatability [116]. Finally, workflows have implicit dependencies on each other, some of which are cyclic and require iterative resolution. Thus, workflows in HEP are highly complex tasks requiring dedicated effort.

In HEP three categories of workflows can be distinguished: a) reconstruction workflows, b) simulation workflows, and c) user workflows. Automated, coordinated workflows reconstruct raw data from the LHC detectors to high-level physics object representations. Similarly automated workflows perform simulations of physics events based on current particle physics theory. The collaboration coordinates both types of workflows and makes the results available to all members of the collaboration. These results form the input to workflows of individual users and workgroups. These workflows are designed for individual analyses, being neither regulated nor coordinated.

Each of these workflow types has different requirements and characteristics. Simulation workflows mainly rely on processing power, whereas reconstruction workflows rely on input data from storage. Both are similarly structured and produce a high volume of data, at the scale of many TB. In contrast, user workflows are not formalised; instead they are customised to the needs of individual analyses. Such workflows span a wide spectrum from processing to input-dominated workflows, and process data at the scale of multiple GB to some TB.

Virtual Overlay Batch Systems

Today, batch jobs of workflows are not directly submitted to CEs at sites. Instead, virtual overlay batch systems are created using the *pilot batch job* paradigm [168, 198]. Pilot batch

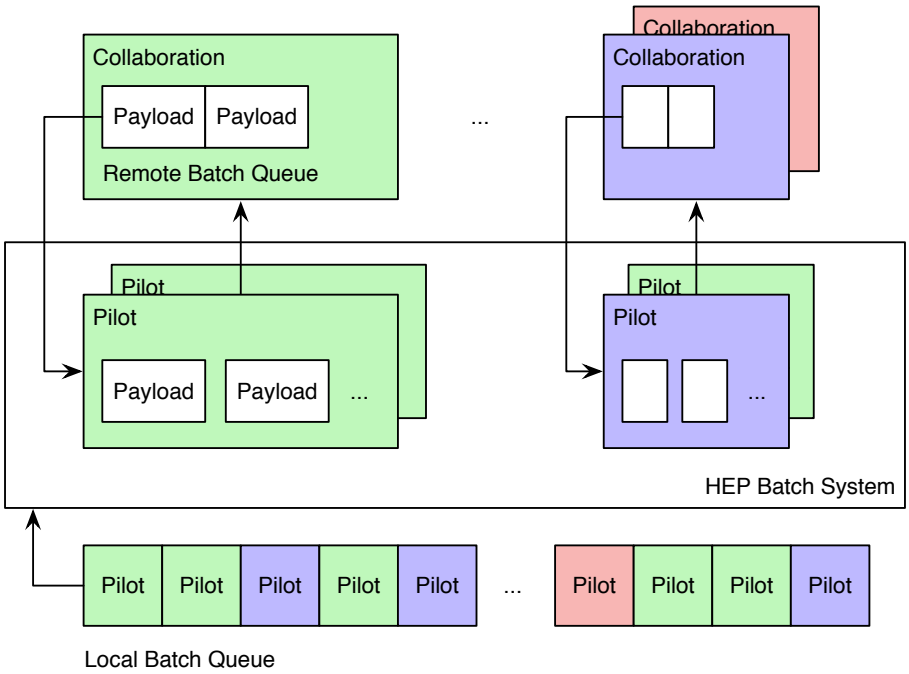


Figure 2.3.: Visualisation of the basic functionality of the virtual overlay batch system paradigm. Pilots are submitted from different collaborations to a batch system. When a pilot starts execution, it connects to the remote batch queue of its collaboration. Payloads from the remote batch queue are sent to the pilot for processing. This allows collaboration-driven scheduling of jobs.

jobs, following called *pilots* for short, are special placeholder batch jobs submitted by each collaboration. Instead of performing work directly, once a pilot starts it connects to a global batch job scheduler of the collaboration. The actual batch jobs, so-called *payloads*, are pushed from the global batch job scheduler to the pilots. The payload then runs in the scope of the pilot by utilising the resources acquired by the pilot. This principle is visualised in Figure 2.3.

On the one hand, collaborations have adopted the pilot batch job paradigm due to its advantages for scheduling and partitioning of resources. Using pilots, collaborations can manage and coordinate resources at multiple sites simultaneously. Users benefit from late-binding features, with direct submission to already acquired resources reducing latencies. On the other hand, sites face less complexity as they do not have to enforce collaboration-internal policies and only a few, technically skilled persons handle batch job submission.

The downside of pilots is an additional layer of encapsulation around payloads. During its lifetime a single pilot can run multiple payloads, potentially even concurrently. Figure 2.4 on the following page shows the distribution of payloads within pilots for the CMS collaboration. With a mean count of 4.82 for payloads and a long tail of the distribution, a pilot must practically always be treated as an arbitrary superposition of payloads. This superposition and encapsulation makes it impossible for sites to monitor single batch jobs, the actual payloads, and subsequently pinpoint causes for errors. Furthermore, the late-binding makes it impossible to optimise batch job scheduling by sites, such as moving simulation workflows to worker nodes with superior processing capacity.

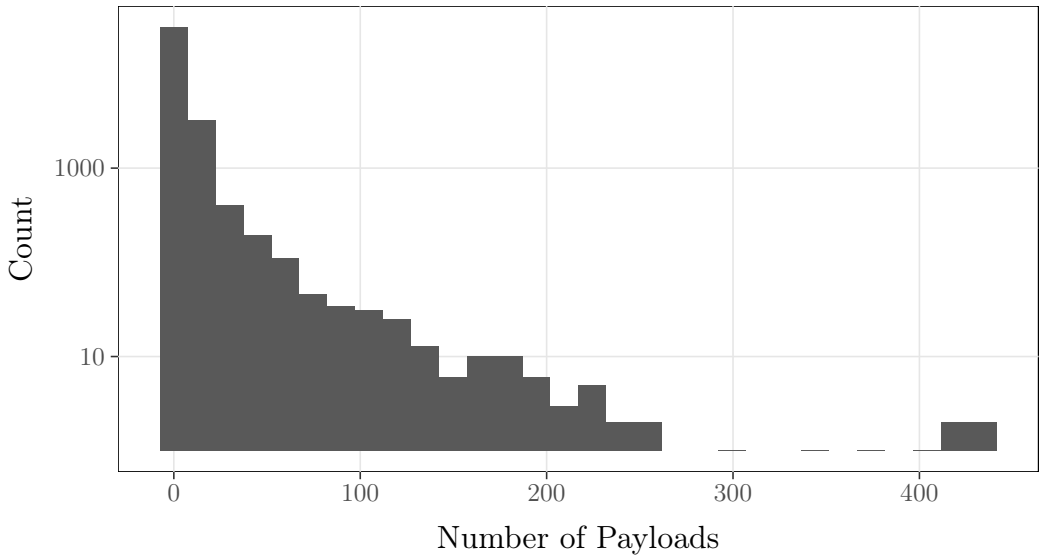


Figure 2.4.: Distribution of payloads in CMS pilots. Each pilot executes an arbitrary number of payloads during its lifetime. Measurements show up to 435 payloads in a single pilot. With an average payload count of 4.82, a CMS pilot can practically always be seen as a superposition of several payloads.

Data Federations

Tentative workflows using remote data access have been shown to improve usability and efficiency of collaborations' computing models [12]. Following the success during the first data taking period of the LHC, this approach was recently adopted as a general strategy. As a result, the concept of data federations was introduced into the HEP computing models. [43]

Data federations provide wide-area access to data via a global namespace. This allows a job running anywhere to request data from a local service. If the data is not available locally, the services can transparently access the data on a remote storage system hosting it. [31, 44, 46, 92]

Transparent file access through data federations notably improves usability and efficiency of workflows [43]. On the one hand, this mechanism offers a fallback for failed file accesses. For example, a job may avert failure if it cannot open single files of a large dataset such as during scheduled or unscheduled maintenance of an on-site storage system. Instead, the fallback mechanism can transparently provide remote access to the dataset. On the other hand, data federations enable integration of additional resource providers, such as diskless computing centres or *opportunistic resources* lacking any on-site storage.

Opportunistic Resources

Within this thesis, we use the term *opportunistic resources* to refer to computing resources, which are not permanently provided by WLCG sites. Instead, opportunistic resources are dynamically acquired to handle shortages and improve overall efficiency. Bird et al. [43] specifically point out that opportunistic resources are important for the current computing

models as upcoming HEP computing and storage requirements cannot be met without external contributions.

Multiple LHC collaborations have started to adopt opportunistic processing resources, such as public and commercial clouds [79, 197], High Performance Computing (HPC) resources [84], or even desktop computing pools [194]. Since these resources are outside of the WLCG infrastructure, their viability relies on transparently interacting with existing infrastructure. Of particular importance is the transparent file access via data federations. Lately, there is an increased use of opportunistic resources to complement existing resources of the WLCG. Especially CPU-intensive but low I/O workflows with little demand for WLCG resources can be outsourced to external resource providers. This utilisation of opportunistic resources has been modelled and showcased through several cloud use cases [110, 228]. In turn, this means, I/O-intensive workflows increasingly exploit WLCG resources.

Network as a Resource

In multi-dimensional environments especially the network is a critical resource. This is especially true when recalling the current mesh-like data access as well as the utilisation of storage federations within the WLCG. As the network is considered an invaluable resource in HEP that enables cost optimisation between networking, storage, and processing [43], special care needs to be taken for network monitoring. In the future, the network will be exploited even more by the different collaborations. One reason is the anticipated increase in data rate that is expected for all four major experiments. Another reason is the aforementioned importance due to cost optimisation by exploiting storage federations. Furthermore the envisaged utilisation of opportunistic resources is important to be considered. On the one hand, usage of opportunistic resources for workflows requiring little I/O will impact the mix of concurrent workflows on the single worker nodes. Thus, network congestion workflows will gain importance. Furthermore, a fair distribution of network resources between the different collaborations should be ensured and guaranteed by the different sites of the WLCG. Consequently, a network monitoring on collaboration or user-level is required. However, the network as a resource is not yet considered part of an service level agreement (SLA).

2.1.4. Virtual Organisations in the WLCG

Given the number of resource providers in the WLCG, coordinating resource sharing for the common purpose to analyse data produced by the LHC is not trivial. Resource providers grant access to different resources, such as CPU, storage, and network. These resources are subject to local policies ensuring SLAs. Resource consumers have their internal policies to aggregate and share resources from multiple resource providers. Both resource providers and consumers want to verify that policies are applied correctly.

To implement authentication, authorisation, and accounting locally for the number of individual users and resources at the scale of the WLCG is not feasible. Thus, the WLCG uses the concept of so-called *Virtual Organisations (VOs)* to ensure scalability and enforce policies for shared resources.

A VO is an abstract entity that groups users, institutions, and resources in the same administrative domain [90]. A VO may also include automated services, acting on behalf of the VO. Based on policies given by VOs and agreements with resource providers, trust is ensured for authorisation. Specifically, resource providers trust each VO to manage its users

Table 2.1.: Example mapping of subgroups of the CMS Virtual Organisation (VO) to local Unix user accounts. As an example for CMS, one can see that each VO can form a complex hierarchical structure with groups and subgroups to distinguish between user privileges. For example, the subgroup *Software Grid Manager* has additional privileges that allows a user to also manage software installations on provided resources.

Name	Description	Group ID	User ID
cms	CMS	5600	14 000–14 199
cmssgm	Software Grid Manager	5600	14 000
cmsprd	Production	5600	14 199
pcms	CMS Pilots	5606	14 900–14 998
dcms	German CMS	5601	32 100–32 298
cmsmcp	Monte Carlo Production	5603	14 800–14 898

and enforce authorisation. This trust model allows services to authenticate, authorise, and perform accounting at the granularity of VOs. Each VO treats its affiliated users uniformly.

VOs may assign specific roles to users, creating fine-grained sub-organisations. However, these are de facto reserved for organisational purposes: For example, roles are assigned to grant additional permissions for superusers or access to resources reserved for particular nationalities.

Within the WLCG, the LHC collaborations are a prime example of VOs representing their members. This membership in a VO enables users to access resources of the WLCG. For this, authorisation and authentication credentials provided by the VO are evaluated by the resource providers. As a result, resource providers may grant individual users access to particular sets of resources. This effectively means a transformation of VO-specific credentials to local ones, such as user identities in a service.

User Mapping and Pool Accounts

Many services in the WLCG rely on resource providers that implement a user management. For example, services analysed in the scope of this thesis are regular programs running on a Unix operating system. The execution on such an operating system requires transforming a user identity at VO level to a single local user account.

Treating users based on VOs implies that service providers act oblivious to individual identities. The full identity of a user, such as its name and institution, are only used to authenticate the user. Authorisation uses VO roles to associate users to different groups; in turn, these groups are mapped to local user accounts.

For groups representing automated services, all identities are usually mapped to a single local user account. Groups of real users cannot map to the same user since this would grant access to personal authentication credentials of all VO members. Instead, such groups are mapped to an array of accounts, called pool accounts.

Pool accounts are a middle ground between abstraction and simplicity for service providers, versus security and protection for users. Mapping to pool accounts is dynamic: The next available account from the account pool is assigned to a new identity whereas unused

accounts are returned to the pool for further assignment. Table 2.1 on the preceding page gives an overview of pre-defined mapping of subgroups of the CMS VO to local Unix user accounts.

By using pool accounts, the individual identity of users is effectively protected from the services. At any moment, each pool account represents a particular user. However, over a period, each account may represent every possible user.

2.2. GridKa Data and Computing Centre

The GridKa data and computing centre is one of the thirteen WLCG Tier 1 computing centres. It provides storage and archival services for HEP data as well as computing infrastructure for large-scale reprocessing and simulation. GridKa supports all four major CERN LHC collaborations: A Large Ion Collider Experiment (ALICE) [196], A Toroidal LHC Apparatus (ATLAS) [62], CMS [172], and Large Hadron Collider beauty (LHCb) [220]. It also provides computing capabilities for several non-LHC HEP experiments, such as BELLE II [50], *BABAR* [161], CDF [36], and Compass [65].

The GridKa is the initial motivation and provides a use case for this thesis. As such, this Section presents an overview of the workflows and resources at GridKa. Unless otherwise cited, information is based on statements from Andreas Petzold, Manager of the GridKa data and computing centre (personal communication, Jan. 9, 2015), Christopher Jung, Representative of the ALICE Collaboration at GridKa (personal communication, Jan. 26, 2016), and Manuel Giffels, Computing and Development Team Leader of the KIT CMS group (personal communication, Feb. 27, 2017).

2.2.1. Resources for HEP Batch Workflows

As a Tier 1, GridKa provides a considerable number of resources for HEP workflows. To support multiple VOs, many of these resources need to be compartmentalised to implement policies and protocols specific to each individual VO. In contrast, the resources provided to several VOs at once are operated without relying on details of the respective workflows.

To actually process the workflows, the most important shared resource for the different VOs is the batch processing. These batch processing resources operated by GridKa appear from the outside as a single pool of resources. Users and VOs submit their workflows to this pool via one of several CEs. Redundant CEs are required for fault tolerance and scalability to handle the number of workflows that are continuously submitted for processing.

As per status beginning of 2017, the GridKa operates around 850 worker nodes in its batch cluster. Each worker node provides up to 24 *job slots*, the maximum number of concurrent batch jobs per node. In total, there are roughly 20 000 job slots, processing up to 2.5 million batch jobs each month [105]. Since the last quarter of 2016, GridKa is in a six month transition phase switching its batch system from Univa® Grid Engine® (UGE) [69] to HTCondor [2].

Batch jobs read input data over network, as worker nodes only provide volatile, local storage. Connection from worker nodes to local SEs, external storage federations, and other sources is realised by a hierarchical network infrastructure. A simplified overview of the network schema is given in Figure 2.5 on the following page. Intra-rack connection relies on 1 Gbit/s data transfer rates. Inter-rack connection as well as connection to local SEs is based on 10 Gbit/s data transfer rates. In addition there are dedicated links to Tier 0 with data transfer rates of up to 100 Gbit/s.

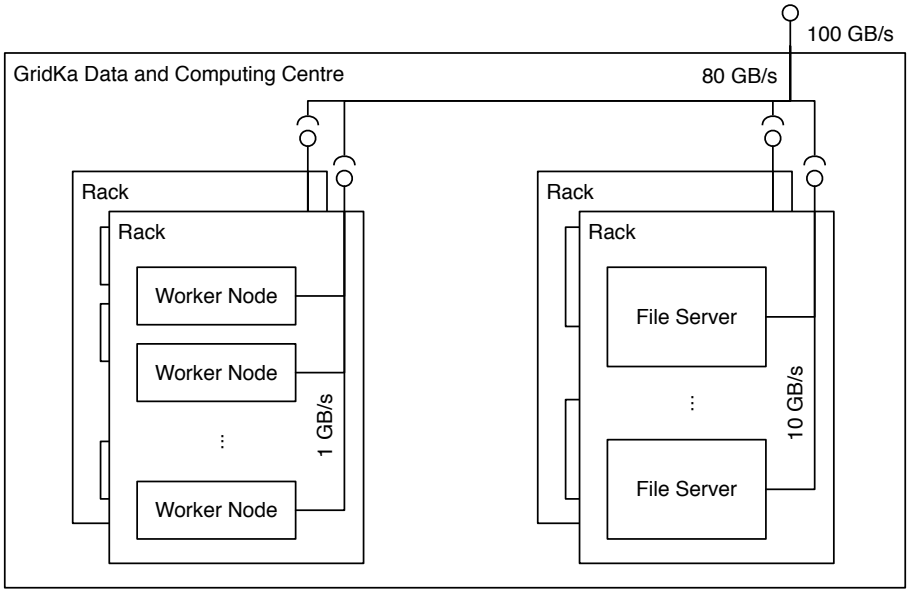


Figure 2.5.: Simplified network diagram of the GridKa data and computing centre. Worker nodes within a rack rely on 1 Gbit/s data transfer rates whereas inter rack communication and communication to file servers allows for 10 Gbit/s up to 80 Gbit/s. The GridKa also provides dedicated links with up to 100 Gbit/s, for example, to Tier 0.

The storage available at GridKa has a volume of several PB spanning dozens of file servers. The provided file servers are partitioned into separate SEs, one per VO. At the end of 2016, the pledged storage volume was 5875 PB for ATLAS, 5250 PB for ALICE, 3300 PB for CMS, and 2500 PB for LHCb [170]. This storage is made available via multiple storage and data access protocols to satisfy the needs of each VO: The SEs of CMS, ATLAS, and LHCb use the dCache storage middleware [91] and provide additional Application Programming Interfaces (APIs) using the XRootD software framework [34]. In contrast, the ALICE SE is purely using the XRootD data access protocol. In addition to the native protocols of dCache and XRootD, VOs also rely on other protocols, such as GridFTP [99], an extension of the File Transfer Protocol (FTP) for data grid projects, or Storage Resource Management (SRM) [203].

2.2.2. Monitoring

This wide range of technologies and infrastructure components makes resource monitoring a complex task. To ensure the quality of service associated with a Tier 1, resource monitoring at GridKa operates at different layers: First, infrastructure monitoring ensures the operability of hardware. Second, service monitoring targets the availability and consistency of provided resources. Finally, VO monitoring validates the fitness to satisfy the requirements of users. As a result, GridKa operates and uses a variety of tools and services for monitoring.

Local Monitoring Frameworks

A number of frameworks and tools are in use at GridKa to collect, visualise, and analyse monitoring data. This variety reflects different use cases and scopes for monitoring. While

there is some overlap in the collected data, each tool serves a different purpose.

Infrastructure monitoring is handled by Cacti® [217] and Ganglia [160], focusing on network and hosts, respectively. These frameworks rely on sensors that continuously collect data of fundamental performance statistics. This includes CPU and memory usage for each host as well as traffic and volume for network interfaces. The collected data is used to manually monitor the state of infrastructure, and identify causes for misbehaviour.

Monitoring of services and their status is handled by Icinga [180]. In addition to passive sensors, its data also comes from active probes which replicate interactions with services. Icinga is an active monitoring system, which is capable of limited analysis of collected data. This serves for automatic notifications of suspected misbehaviour. Icinga is a key component to ensure service availability, acting as an early warning system and alerting on-call duty for severe issues.

In addition, many services internally collect status information. This information can be accessed manually to verify suspected issues and inefficiencies. To simplify the exploration and correlation of such data, key features are stored in databases. For example, GridKa collects performance characteristics of each batch job in its batch system. This data can be dynamically analysed using Grafana [169] and Kibana [81], focusing on time series and correlations, respectively.

Experiment Dashboard

Each VO deploys its own monitoring, separate of local site efforts. This monitoring relies on sensors which are directly integrated into VO middleware and frameworks. As such, VOs are capable of monitoring individual operations and can relate this to the context of workflows. While this monitoring is more fine-grained than local site monitoring, its scope is limited to operations fully under the control of a VO.

Many monitoring sensors have been developed and are in use by the individual VOs [30, 60, 195]. These sensors are specific to particular workflow management systems, data management systems, or application frameworks. To combine information across VOs, the Experiment Dashboard [14] allows to aggregate, compose, and visualise statistics independent from underlying monitoring technologies. While all VOs publish data to the Experiment Dashboard, the scope and volume of data varies. Depending on the VO, monitoring data is available on transfers, processing of workflows, and infrastructure.

The CMS *Task monitor* provides real-time and historical status information on all started CMS jobs. This combines information from the workflow submission system, the payloads performing the work and the pilots representing acquired resources. However, it relies on payloads actively reporting data – failures during payload startup and misconfiguration prevent tracking of jobs. The ATLAS *data management monitor* provides a global view of data transfers between WLCG sites. The heterogeneity of data transfer technologies for the different VOs requires a cross-VO and cross-technology view of data transfers within the WLCG infrastructure. The data management monitoring application provides a unified view on results gathered from coexisting technologies[15]. Finally, the *infrastructure monitor* summarises results based on so-called *Service Availability Monitoring (SAM)* tests.

Service Availability Monitoring

The SAM framework is a distributed, global monitoring infrastructure responsible for monitoring the WLCG and its resources [13]. It is based on probing functionality of services

and simulating individual steps of user and collaboration workflows. SAM is used both as an early warning system and to validate the SLAs.

The monitoring framework comprises several functionality: submission of monitoring probes, gathering of probe results, processing of monitoring data, and evaluation of results regarding service status, availability, usability, and reliability. For example, SAM tests of processing resources submit probes to CEs to validate the submission process. Once deployed on worker nodes, the probes check the availability of VO-specific dependencies and the usability of the infrastructure in general.

Results of such probing express the availability of site resources to fulfil the SLA for specific VOs. However, SAM tests are not suitable in identifying causes for issues beyond fundamental service unavailability. Being deployed like regular resource consumers, SAM probes are oblivious to the underlying infrastructure.

2.2.3. Tracking of Batch Job Behaviour

By design, utilisation of batch processing resources is only partially under the control of sites. Each batch job can freely use the fraction of resources it is allocated to on a worker node. In addition, allocation is limited to resources that can be easily controlled for each batch job. As a result, monitoring and tracking of batch jobs to identify misbehaviour is critical to ensure the efficiency and availability of processing resources.

Resource Allocation for Batch Jobs

The GridKa batch system allocates both CPU cores and memory to batch jobs. During the runtime of a batch job, the availability of these resources is guaranteed. This allocation is enforced as a soft limit: Batch jobs can temporarily exceed their limit if resources are unused. However, scheduling of new batch jobs treats all limits as fixed.

This allocation approach ensures that running batch jobs cannot over-utilise resources. A running batch job cannot be starved of resources by other batch jobs. Yet, misbehaving batch jobs can reduce throughput by underutilising resources. A batch job requesting and thus blocking more resources than needed prevents new batch jobs from starting. In addition, allocation is performed only with a limited resolution on both resources and time: Batch jobs exiting immediately after starting, either through a defect or dynamic configuration, block resources for much longer than the runtime of the batch job.

HEP batch jobs implicitly rely on a range of other resources which are not allocated by the batch system. Instead, these resources are available to all processes on worker nodes, and thus shared by all batch jobs. The allocation of these resources is not handled at the scope of the batch system, but by the operating system at the scope of each worker node.

Network in particular is an unallocated but critical resource for HEP batch jobs. All batch jobs rely on network to read input from and write output to SEs. As such, congestion of network usage has direct results on batch job throughput and thus efficiency. Since network is shared at multiple levels, primarily on the worker nodes and file servers, few misbehaving batch jobs can impact a high number of regular batch jobs. While symptoms can be detected easily, the initial cause is not visible to monitoring.

Example 2.1. Consider a group of batch jobs failing to open files locally. Thus, each batch job in this group falls back to reading data remotely from other sites. In this case, monitoring can detect a firewall being heavily loaded, and incoming network being at full capacity. Each individual worker node, running only a handful of misbehaving batch

jobs, would not be identified as anomalous by monitoring. Instead, pinpointing individual, misbehaving batch jobs requires correlating multiple sources of monitoring data. Causes must be isolated based on exclusion and cannot be located directly.

This issue described in the Example 2.1 on the preceding page arises as the granularity of monitoring and that of anomalies is mismatched: the former targets elements of the infrastructure, while the latter are small elements distributed over the infrastructure. For example, an individual batch job makes up only a part of the activity on a worker node. Yet, multiple batch jobs may depend on the same shared resource, with each batch job running on a different worker node.

Detection of Batch Job Misbehaviour

Before an administrator can take action against misbehaving batch jobs in the batch system, the misbehaviour itself must be detected. This can be done by automated alarms, manual validation of passive monitoring, or, in the worst case, by user reports. We use the term *detection point* for the trigger that alerts the administrator about the misbehaviour. For example, an automated alarm for the firewall being saturated by ingoing network traffic is a detection point.

The granularity of available monitoring tools (see Section 2.2.2 on page 16) decouples the detected issue from the underlying cause. For example, the detected traffic saturating the firewall is actually a superposition of thousands of network connections. To find the root cause of an issue, administrators need to investigate all involved components of the system.

In the case of misbehaving batch jobs, the issue is further complicated by the high concurrency in the batch systems. Errors in workflows possibly affect all jobs of the workflow at once; if this causes network saturation, only an excess of connections from the entire batch farm can be detected. Since each worker node runs dozens of jobs in parallel, local effects of misbehaving batch jobs can be obscured by regular jobs. The latter is further complicated by pilots, which make the identification of individual payloads and their effects impossible.

Example 2.2. Recall the Example 2.1 on the preceding page where a specific group of batch jobs exclusively accessed data from remote sites. In this example, the behaviour is caused by a software update of an analysis framework implementing transparent access to files via data federations (see data federations in Section 2.1.3 on page 12). This update corrupts the evaluation of local file availability resulting in remote file access by default. Thus, software building upon the changed framework exclusively reads from external SEs. Monitoring can only detect the increased traffic on the firewall. Manual investigation of network connections can reveal an increase in communication from worker nodes to remote servers. Only if remote servers belong to a specific VO this is sufficient to identify the general group of jobs. Otherwise, the number of jobs per VO on each worker nodes must be correlated to local traffic. If neither approach yields results, data from external monitoring, such as VO dashboard statistics on file transfers, is required. If a VO is identified as the cause of the problem, it is informed about the issue. Meanwhile, the site can only limit the number of *all* batch jobs of the affected VO to protect site availability.

This thesis focuses on batch job misbehaviour with a possibly negative impact on running batch jobs. The goal is to move from a reactive mechanism described above to detecting misbehaviour in near real-time. Thus, the focus is to enable a detection point at runtime of

batch jobs in order to minimise the impact on other batch jobs. Ideally, false alarms are minimised to avoid the high cost of manual investigation. However, false negatives also have to be minimised as these may result in missed activity while costs are incalculable.

Without loss of generality, we focus the analysis of batch jobs to an analysis of network traffic. Based on the trend of the usage of opportunistic resources (compare opportunistic computing in Section 2.1.3 on page 12), especially the analysis of network traffic utilisation becomes more important. In addition, the network resource is a resource that is shared by many batch jobs. Thus, a fair allocation needs to be guaranteed as batch jobs interfere each other [39]. For memory and CPU resources this is already handled by the batch system, but not for network.

2.3. Complexities in High Energy Physics Batch Systems

User-centric monitoring in a HEP batch system is much more challenging than in a controlled environment or on a solitary machine. There is a multitude of concurrent applications, user inputs vary in non-obvious ways, and applications are frequently re-written and updated (compare Section 2.1.3 on page 9). In the following, we summarise challenges in HEP batch systems and data centres preventing the use of established methodologies for user-centric monitoring of batch jobs. These challenges outline the bounding conditions of the exemplary use case in this thesis. While the discussion is based on the view of HEP batch systems only, challenges also partially apply to other batch systems and data centres.

Heterogeneous Hardware The increasing data volume of HEP collaborations requires a steady growth of resources pledged by WLCG sites. This results in incremental, horizontal scaling of provided resources. As a result, new processing and storage hardware is regularly added to the pool of already existing hardware. Reflecting the technological progression and the purchase of the most financially efficient hardware, extensions result in a highly heterogeneous environment. This requires tools that are not dependent on specific hardware capabilities.

High Utilisation of Worker Nodes In batch systems focused on high throughput, a high utilisation of processing resources is desirable. This means a high level of concurrency, as each worker node hosts several dozen, independent batch jobs in parallel. Each batch job in turn is expected to fully utilise its allocated resources. Worker nodes are thus expected to work at full capacity, making excessive resource usage non-trivial to detect. Even disentangling a small number of applications can be difficult [164].

Heterogeneous Job Mix With batch jobs having diverse resource requirements, those demands must be mixed by distributing each batch job class equally. This optimises resource usage by balancing requirements [42, 48]. However, this balancing complicates monitoring as the total host utilisation is purposely decoupled for each individual batch job. Monitoring data used to optimise the utilisation of specific resources, for example opportunistic resources, must adequately reflect the impact of individual batch jobs [39, 122, 127].

Ambiguous Job Definition Based on discussion in Section 2.1.3 on page 9 the apparently trivial definition of a single user job is very complex. Due to the pilot batch job paradigm, the jobs deployed by end users are not equivalent with the batch jobs deployed by sites. Instead, individual payloads within a pilot running as a batch job

are the actual jobs that must be considered for user-centric monitoring. This requires the extraction of payloads from batch jobs as part of the monitoring.

Ambiguous User Definition While users are authenticated individually, their identity is not propagated to batch jobs (see Section 2.1.4 on page 14). Furthermore, each payload running inside a pilot performs authentication at runtime, independent of the site's batch system. The information available from processes and the site's batch system provides only a rough categorisation on VO-level. The real identity of a user might be collected by evaluating the user mapping at runtime. However, this is not a system-independent operation and is in addition prevented for reasons of privacy and confidentiality. Thus, specific information about the shared identity and intent of batch jobs cannot directly be derived.

Range of Input Characteristics HEP batch jobs heavily rely on trivial parallelisation by executing the same workflow with different inputs (compare Section 2.1.3 on page 10). While batch jobs of the same workflow are logically comparable, the differing input can have a significant influence on performance and duration. Monitoring must therefore allow to identify batch jobs not just by performance characteristics, but also general features.

Unknown Optimal Behaviour To derive the underlying workflow of payloads is a non-trivial task. In a production batch system it is not feasible to isolate a specific payload from other processes. This would require coordination of both, sites and collaborations, and the circumvention of scheduling policies. Due to natural interference between batch jobs, an isolated job is likely to exhibit different behaviour. As a result, it is practically impossible to determine the optimal behaviour of individual batch jobs under realistic conditions. This excludes the possibility of deriving an objective baseline for optimal behaviour of payloads.

Limited Monitoring Capabilities As the monitoring of batch jobs needs to be performed in a production environment, a low performance overhead is crucial. The main focus of batch system resources is on processing of batch jobs, so excess resource consumption would interfere with both batch job throughput and measurements. Both allocated resources such as CPU and memory, but also unallocated resources such as network may not be consumed substantially. Overall, the deployment cost of monitoring must be negligible compared to normal operations.

A user-centric monitoring of batch jobs should address the different aforementioned challenges in order to be feasible for deployment in HEP batch systems. However, an actual monitoring solution must not only provide characteristic information, but allow the assessment of information in the scope of distributed workflows. The challenges and scope call for a dedicated monitoring tool to enable the online analysis of payloads.

3. Monitoring of High Energy Physics Batch Jobs

User-centric monitoring in batch systems offers great potential to optimise the detection point of misbehaviour in batch systems (compare Section 2.2.3 on page 19 for a discussion of possibilities). Especially if batch jobs are identical instances forming a single workflow, early detection of misbehaviour allows preventive action against later batch jobs of the workflow. Consequently, optimisation of the detection point can result in a reduction of disruption of other batch jobs. However, user-centric monitoring is a highly complex task, especially for systems built out of distributed components (compare Section 2.3 on page 20 for an overview of specific challenges for HEP batch systems).

The approach to user-centric monitoring and some of the results discussed in this Chapter have originally been published in Kuehn et al. [139–141]. All key arguments and information required for later Chapters are presented here.

3.1. Related Work

There are many approaches and tools for monitoring in batch systems and similar environments. However, they are typically specialised for a particular layer or application. To the best of our knowledge, existing solutions are not ideal for monitoring applications in HEP batch systems. Most importantly, the granularity to monitor individual payloads in a distributed system is not provided by any of the available methods.

In the following, classes of existing approaches and tools are outlined and evaluated for their applicability. Based on this, we present an approach to user-centric monitoring, focused on the requirements in HEP batch systems. However, the approach can also be applied in any Unix-based system where a hierarchical view on processes is relevant.

3.1.1. Taxonomy to Host-Based Monitoring

To consistently monitor batch jobs, data must be collected at a scope encompassing the entire batch job runtime. At the same time, this scope must be small enough to minimise the collection of unrelated data. Since HEP batch jobs do not interact with each other, this is trivially the pilot or payload itself. However, in our use case, both users and their workflows are outside the control of monitoring tools. The next available scope is the batch system running the batch job; yet, this lacks control over vital resources (as outlined in Section 2.2.3 on page 18). Finally, the host as a whole is guaranteed to control all resources a batch job can directly access. Thus, we focus on host-based monitoring solutions with the goal to monitor individual payloads.

There has been extensive research on host-based monitoring approaches appropriate for user-centric monitoring [3, 6, 127, 200, 222]. These approaches have different trade-offs in requirements, performance impact, and granularity. To provide an overview of this

field, a taxonomy to basic functionality is presented in the following. Table 3.1 on page 27 summarises characteristics of the available classes of host-based monitoring.

Software Instrumentation

Monitoring based on *software instrumentation* modifies the target application by injecting monitoring instructions, the instrumentation code. The primary goal is to record internal application state and behaviour, for example for program analysis, debugging, performance optimisation, and virtualisation purposes [128, 213, 242]. As an example, instrumentation code can be added at the entry and exit of each function call to log its name, arguments, and return value. Thus, software instrumentation enables a fine-grained intra-process analysis on instruction-level.

The granularity of software instrumentation comes at a high cost: Software instrumentation purposely modifies the target application, potentially slowing it down with additional instructions or negatively affecting the cache behaviour. This can obscure monitoring data by preventing the execution of critical code sections, or avoid serious boundary conditions. For example, the slowdown by software instrumentation may significantly reduce data throughput, preventing the collection of data on network saturation. Thus, disentangling normal and anomalous behaviour can become impossible without prior knowledge.

Several VOs employ moderate software instrumentation compiled into their analysis frameworks for monitoring [60]. However, site administrators cannot similarly recompile user software to add site-specific instrumentation.

Static Instrumentation Static instrumentation techniques insert instrumentation code to the target application during or after compilation but before execution. For example, the LLVM framework can be utilised to automatically add instrumentation code at compile-time [146]. Nowadays, static instrumentation techniques are mainly used in the fields of performance analysis, error detection, and software quality assurance and testing [173, 225]. However, static instrumentation has some severe limitations [158]: a) Static approaches modify the software executable and thus require an extra step before execution of the program. b) Static instrumentation can only cover statically linked resources. The tracking of dynamically linked code, shared libraries, or dynamically generated code is not possible. c) Furthermore, code and data can be mixed in executables, for example in static libraries; Static instrumentation cannot reliably distinguish data from code in such cases.

Dynamic Instrumentation Dynamic instrumentation techniques modify target applications such as compiled executable binaries or bytecode dynamically during execution. Thus, they do not require a separate step to add instrumentation code but alter the target application for each execution. During execution, dynamic instrumentation tools insert themselves between the target application and the executing host machine. For example, dynamic instrumentation tools such as Pin [158] use the `ptrace` [133] system call to gain control of the target application. This way, dynamic instrumentation tools can monitor the execution at instruction-level to inject instrumentation code as needed.

The most significant advantage of dynamic instrumentation is that any executable can be tracked and analysed without requiring development efforts. Thus, dynamic instrumentation supports to instrument dynamically generated or linked code as well as shared libraries. However, the execution overhead of a target application is increased significantly when dynamic instrumentation is performed at runtime [128, 205].

Whole-System Monitoring

Whole-system monitoring approaches treat the monitored application as a black box. By tracking the interactions of the target application with the operating system, characteristics of the application are inferred based on the resources it uses and generates, for example, files, sockets, processes, or peripherals. Such interactions are usually performed via system calls such as `open`, `read`, `write`, `socket`, or `ioctl`. These system calls can be monitored with tools such as `strace` [134].

However, this method comes at a high cost as it pauses the process before and after each system call that is traced. Instrumenting the whole system incurs significant performance and analysis overhead. To efficiently obtain information on the whole system, many tools modify the OS kernel or use a specific kernel module [28, 179]. These approaches reduce overhead at the expense of maintainability and potentially stability.

System Emulation and Virtualisation

To avoid modifying the OS for whole-system monitoring, other approaches are based on emulation or virtualisation of the host system. By utilising dynamic software instrumentation on the emulated host system [78], fine-grained information can be collected. This approach is utilised in record and replay platforms to capture low-level information; the volume and granularity of data require further analyses to be performed offline [204, 235]. Also, emulator-based recordings result in a significant slowdown which prevents realtime analyses and deployment in production environments. Virtualisation-based approaches, however, show less significant overhead [135].

Hierarchical System Monitoring

The Linux operating system allows subdividing resources of a host into smaller subsets. These control groups, or commonly cgroups [132], enable a hierarchical system monitoring. When using cgroups, a fraction of CPU, memory, disk I/O, or network resources are isolated and assigned to a group of processes. This enables regular monitoring of system usage for the specific group, and low overhead compared to emulated or virtualised systems. However, without prior knowledge, each cgroup must treat its processes as a single entity consuming resources. Furthermore, monitoring network traffic requires the creation of multiple virtual network interfaces [176].

Hardware Assistance

The last approach we distinguish uses hardware features to capture information about software execution. For example, the latest Intel CPU supports the Processor Trace feature [114]. These hardware features allow precise control flow tracing of a process without any system modification. Furthermore, the performance perturbation to the software is negligible. This allows portability of appropriate approaches and directly makes them available on many platforms. However, information is gathered at a very low level, and many CPUs do not support such features yet.

3.1.2. User-Centric Monitoring

There are several tools and suites to collect information on jobs in environments similar to HEP batch systems. Each builds on one or a combination of the monitoring approaches

mentioned above, but aggregates and processes data differently. As a broad classification, tools are distinguished by their balance of information granularity, deployment scope, and autonomy.

Application profiling and monitoring tools such as Vampir [136] give insight into performance parameters of a particular application. However, their overhead is too high for continuous monitoring. Therefore, they are commonly used to tune individually selected applications explicitly. This is not feasible for administrators of batch systems running thousands of batch jobs in parallel.

Approaches such as Lightweight Distributed Metric Service (LDMS) [4], Holistic Performance System analysis (HOPSA) [162], and Google-Wide Profiling (GWP) [185] explicitly focus on scalability for production environments while providing continuous sampling-based performance data for applications. The focus on scalability allows a large scale deployment in computing farms, enabling cross-correlation analysis for different hosts on application-level. However, existing approaches are not applicable to HEP environments.

In Agelastos et al. [3] the authors utilise LDMS to enable job performance evaluation on application-specific pre-calculated scoring methods. However, the approach is geared towards HPC resources, exploiting the monitoring of entire worker nodes as each job fully utilises several worker nodes. In contrast, our use case requires resolving multiple individual jobs running concurrently at the same worker node.

The approach proposed by HOPSA enables the analysis of co-located jobs by creating an integrated diagnostic infrastructure on application and system-level. This uses a multitude of integrated monitoring tools and components collecting multiple types of data that may vary in frequency, subject, and granularity. A significant overhead is required to analyse inter-node or intra-node dependencies.

The approach of GWP also employs function-level monitoring. This enables analysis and profiling of function-level call graphs of an application. The capability to provide profiling data in a hierarchical layout allows users to focus on specific parts of an application, as stressed by the authors. This mimics our need to monitor payloads as parts of a pilot. However, this type of analyses is not automated and must be performed manually by users in an explorative fashion. Furthermore, the sampling approach does not facilitate analysis of temporal dependency between function-level calls that is required for our approach to learn underlying workflows.

3.1.3. Data Collection in Production Systems

A user-centric monitoring approach for HEP batch systems needs to satisfy several boundary conditions: First, the monitoring approach must be suitable to work on real applications. These may be large, multi-threaded, composed of several processes and executed in parallel with other, different applications. Second, the approach must be suitable for deployment in a production environment, including unmodified runtime environments and no operating system extensions. Isolated testing environments or extensive integration efforts would limit the applicability and portability. Third, the overhead from monitoring should be kept to a minimum. Data collection must not distort job execution to remain generalisable, and must not notably reduce job throughput. Finally, the monitoring setup should allow for an estimate of misbehaviour at runtime.

Each of the various monitoring approaches fulfils different requirements (see Table 3.1 on the facing page). An adequate solution must balance individual needs; Most techniques trade one feature against another.

Table 3.1.: Overview of host-based monitoring approaches. Each class of monitoring approaches is evaluated with regard to its characteristics. The notation from −− to ++ gives a relative measure on the suitability regarding the given characteristic. Characteristics on *integration* and *administration* costs reflect the required short- and longterm effort. The *overhead* represents CPU and memory consumption of the monitoring itself as well as slowdown of the target application. *Granularity* compares the amount of information tracked. Finally, *coverage* gives information on the subject of monitoring.

	Cost				
Class	Integration	Administration	Performance[1]	Granularity	Coverage
Software Instrumentation					
Static	−−	++	−	+	process
Dynamic	+	++	−−	++	process and children
Whole-System Monitoring					
System Tools	++	++	−−	+	process and children
Kernel Modules	−	−−	+	+	depends on approach
Emulation / Virtualisation	+	−	−− / −	++	process and children
Hierarchical System Monitoring	+	−	++	+	process and children
Hardware Assistance	−	+	++	+	process and children

[1] includes memory and CPU overhead as well as slowdown of applications.

Especially for whole-system monitoring, the runtime overhead can significantly be improved at the cost of higher integration effort [28, 179]. Collecting data directly with specific kernel modules or kernel modifications ensures little performance penalty. However, portability and maintainability are severely limited. Interaction with the kernel limits compatibility to individual kernel versions. Furthermore, both development and maintenance require an in-depth understanding of the kernel. As such, a solution with indirect kernel interaction is preferable.

Software instrumentation offers the best granularity since it has direct access to the application. However, static instrumentation is not feasible to apply without control over user application. While dynamic software instrumentation can be deployed externally, it incurs a high overhead. The slowdown of target applications is significant [128, 205], making any recorded data unsuitable to reason about normal batch job execution. Software instrumentation may be suitable for users and collaborations but is unsuitable for continuous monitoring. However, the general approach of monitoring individual actions that constitute an application is promising.

System emulation and virtualisation offer reduced overhead by uncoupling data collection and analysis. Related work proposes to record the execution of applications and deterministically replay it when required [118]. This mitigates some of the overhead incurred by instrumentation. Usually, those systems rely on an offline replay of records without further optimisation of overhead. The delay between record and replay renders this approach unfeasible for an online analysis of batch jobs.

Hierarchical system monitoring via cgroups induces less overhead compared to emulation or virtualisation-based approaches. Batch systems allow deploying batch jobs with cgroups for resource allocation and monitoring [96]. This is restricted to the granularity of batch jobs, and cannot differentiate between payloads.

Following this analysis, we base our user-centric monitoring for batch systems on data sources close to the kernel: Kernel data is accessed from user space using pre-existing tools and APIs. This preserves most performance advantages of using kernel data while avoiding the complexities of direct kernel interaction. However, we strive for hierarchical data mimicking the call graphs available from software instrumentation. As HEP applications consist of an extensive process hierarchy, we assume that this allows for similar analyses as on call graphs.

3.2. Methodology to User-Centric Monitoring

For our work, user-centric monitoring must fulfil two requirements: First, it must be suitable for long-term deployment in production environments of HEP batch systems. This is a constraint to use monitoring in a meaningful way. Second, it must provide data suitable to identify and reason about batch jobs of workflows. This is the motivation for introducing a new monitoring approach in the first place.

As such, the approach can be roughly divided into two views: The implementation of *how* data is collected, and the purposeful selection for *which* data is collected. Notably, we exclude the analysis and interpretation of data at this point.

Based on our review of host-based monitoring methods, we use an approach that indirectly interacts with the kernel: We rely on standard system calls, passively exposed information, and existing stable libraries. Thus, the solution does not require any kernel modifications or kernel modules. This ensures portability and stability for any reasonably modern

environment. Furthermore, the overhead of data collection is very close to the optimal. Overall, this renders the approach feasible for continuous monitoring of batch jobs.

The utilisation of indirect kernel interfaces enables a high granularity of monitoring. It provides per-process information such as process start and exit, resource utilisation of CPU and memory, and external resources such as files and network sockets. Thus, the dependencies between individual processes can be derived from different levels. For example, data shared between processes via files is an active research topic in the field of automated provenance detection [118, 185, 213]. For HEP in specific, network is a critical resource (see discussion in Section 2.1.3 on page 12). Consequently, our proposed approach exemplarily focuses on network I/O for individual processes. We explicitly exclude the consideration of dependency graphs based on data flows at file level. Instead, we refer the interested reader to available approaches and results [118, 185, 213].

3.2.1. Towards Modelling of Workflows

The data available from the kernel does not directly translate to jobs by users. As such, a model of jobs is needed to relate them to kernel information. The execution of workflows involves a stack of concepts (see Section 2.1.3 on page 10), necessitating a precise definition of elements:

Definition 3.1 (Workflow, job, payload, and pilot)**.** Analysis efforts of a VO are composed of *workflows*. Each workflow implements a deterministic algorithm to transform data from an input dataset. A workflow consists of many *jobs*, each applying the algorithm on a distinct subset of the input dataset.

A job is an abstract description of an algorithm and target data. It is implemented as a *payload*, which is an executable application and configuration defining its input.

To process workflows, a *pilot* runs as a batch job in a batch system to acquire resources. Each pilot, in turn, executes one or multiple payloads.

The relationship between these concepts is denoted in Figure 3.1 on the next page. Indirectly the different concepts create two connected, inverted hierarchies: From the view of a batch system, each pilot is a group of processes to which each payload is a group of child processes. From the view of analysis, each job is a child of the underlying workflow. Based on this model, our goal is to identify payloads in a pilot process hierarchy and link the corresponding, logical job to the workflow.

It is inherently difficult to infer the workflow without having information on its semantics. It can be composed of several algorithms and tasks that depend on each other or even have circular dependencies. However, we can exploit that each job is the execution of the same algorithm, differing only in input. Thus, we assume that jobs of the same workflow behave similarly. By statistically analysing characteristics of payloads, fundamental characteristics of workflows can be derived.

Thus, significant characteristics of payloads need to be considered: First, the features that are fundamentally similar for all payloads that are based on the same workflow. Thus, knowledge about features that is characteristic for specific workflows is required to identify similarities. Second, the granularity of at which features are observed is of vital importance. Monitoring must provide sufficient granularity to allow differentiating payloads of different but similar workflows. Finally, the sampling rate also needs to be considered. A high temporal resolution enables near real-time analysis by reducing the latency between data

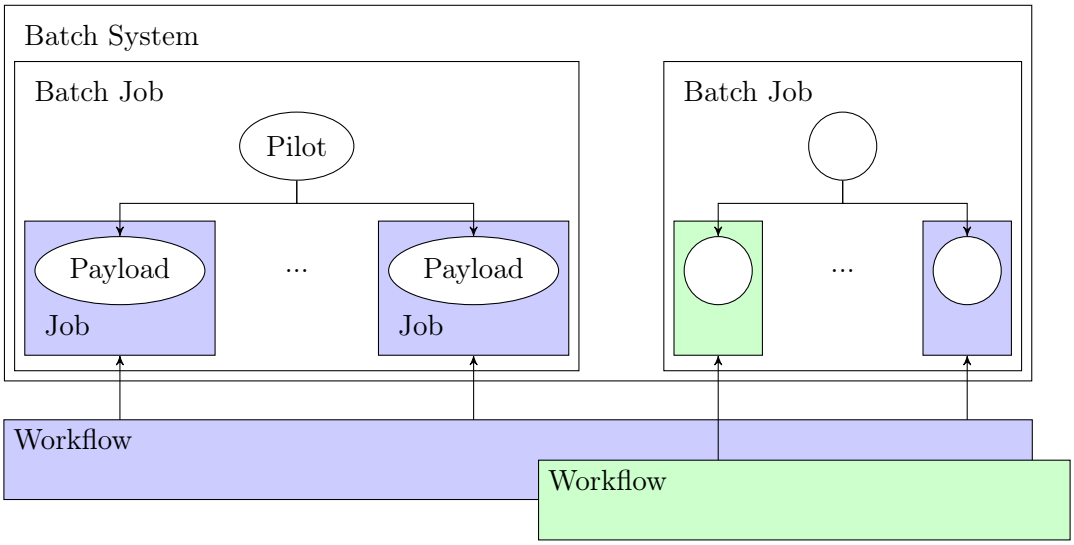


Figure 3.1.: Overview of classification of workflows, jobs, batch jobs, pilots, and payloads. While workflows and each individual job of the same workflow algorithmically and organisationally describe the same tasks, the payload is the actual realisation of the job. Thus, a payload represents a job in the batch system. Due to the pilot paradigm, several payloads are encapsulated within a pilot. One pilot does not necessarily encapsulate payloads belonging to only one workflow. Finally, the batch job encapsulates the pilot and its payloads.

generation and collection. However, each of these characteristics must be balanced against minimising the volume of data to keep it manageable.

Characteristic Data

The identification of workflows can be divided into two tasks: First, individual payloads must be isolated from their pilot. Second, each payload must be identified with regards to other payloads. This translates directly to two types of data that monitoring should provide.

The pilots used in the WLCG directly execute each payload as a separate process. This forms a hierarchy of groups of processes: The root is the group of processes making up the pilot itself. Leaf processes of the pilot spawn the root processes of each payload. From each payload root process, a separate group of processes is started, making up the actual payload. As such, monitoring must provide information on the *process hierarchy* for each batch job – that is, the identity and relation of pilot and payload processes.

The primary classification of HEP workflows is their use of input data concerning volume and throughput (see Section 2.1.3 on page 10): Jobs with little input data are usually classified as simulation workflows. Jobs with much input data can be classified as reconstruction workflows. As input data is not stored locally on processing nodes, we assume that network usage of payloads is characteristic for different workflows. Thus, monitoring must also provide *network usage* at payload granularity – due to how payloads are isolated; this means the network characteristics of processes.

In addition to identifying workflows, network characteristics also constitute viable features of interest. On the one hand, it provides predictable monitoring of this critical resource.

This is especially relevant concerning opportunistic resources, which are feasible only for workflows with little input data. On the other hand, it is an indicator that processes are performing viable work.

Data Granularity and Sampling

Data characterising jobs can be collected at various granularity. For example, the hierarchy of pilot and payloads can be expressed via groups of processes, individual processes, or the state of each of these processes. Similarly, network traffic can be tracked for example in total, per interface, subnet or address. Thus, the description of a job via monitoring can in principle be arbitrarily complex.

Increased complexity enables a detailed view and analysis of jobs. However, it comes at the cost of increased memory, storage, and processing resources to handle the additional data. Also, a more fine-grained granularity is subject to noise, whereas a coarse granularity may miss significant information.

We differentiate between two types of granularity: Data granularity describes the level of distinct data provided by monitoring. Roughly speaking, it is the number of different features that are tracked. Temporal granularity describes the detail of consecutive data. For actively collected data, this can be influenced via the sampling rate. For event-based data, the event rate is generally outside our control; only by excluding certain events can granularity be controlled.

3.2.2. Monitoring Workflow Features

To realise a monitoring system for HEP workflows, the requirements on data types and granularity must be aligned. Hierarchical information must be sufficient to split payloads from their pilots. Network information must allow identifying payloads of workflows with similar process hierarchies. Both types of information must be compatible with each other and restricted to a manageable volume.

The border between payload and pilot processes is vague. On the contrary, each pilot implementation uses a different sequence of processes to prepare and start a payload. On the other hand, each payload again wraps the actual job into a layer of processes to prepare and monitor it. This makes it important to collect the entire process hierarchy to precisely separate the real payload.

This requirement aligns well with process data exposed by the Linux kernel. Data can be collected in an event-driven fashion: Separate events can be defined for the start and end of a process. This sets the process data sampling rate to the frequency of process events. Information available on processes allows relating them to their position in the process hierarchy.

Following this, collected network information must be suitable for association with individual processes. However, network information is available at a very detailed granularity: network events, the individual packets, can reach a volume of several hundred thousand per second even on a 1 Gbit/s interface. Thus, information must be translated to a higher level of abstraction.

A sampling approach is suitable to link the high granularity network information to process data. Individual network events can be aggregated to connections based on their source and destination addresses. Without prior knowledge about remote addresses, one can locally distinguish between network interfaces – in the case of GridKa, internal and

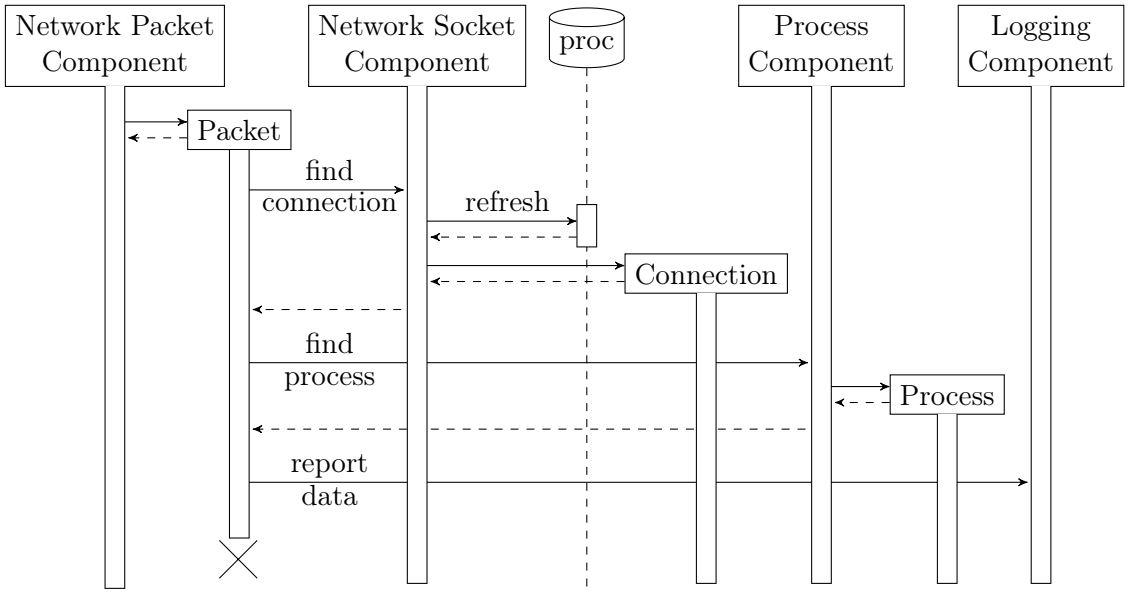


Figure 3.2.: Visualisation of interaction between the different components of BPNetMon. Three main components are differentiated: the Network Component that tracks data from network packets and connections, the Process Component that collects events from processes within the operating system as well as the Logging Component.

external connections can be distinguished. Besides, ingoing and outgoing data rates can be separated without prior knowledge.

3.2.3. User-Centric Monitoring Sensor

As part of this thesis, a sensor for user-centric monitoring in HEP batch systems has been developed. The following gives an overview of this tool, as required for later Sections describing the analysis of collected data. An in-depth description of algorithms, data structures, and optimisations can be found in Appendix A.2 on page 154.

Our user-centric monitoring for HEP batch jobs collects information to relate Unix processes and network traffic. This approach is based on the general idea of the OpenSource tool NetHogs [83]. NetHogs groups bandwidth by process instead of breaking traffic down per protocol or subnet. However, as NetHogs is not suitable for continuous monitoring, it cannot be deployed for long-term, large-scale monitoring. Instead, we have developed a tool suitable for continuous, user-centric monitoring, called Batch Process Network Monitor (BPNetMon).

Our tool can be divided into two main tasks: the monitoring, and the logging task to provide the collected data to consumers. This facilitates a modular design for future extensibility – in the following, we restrict the description to components used for this thesis. There are three monitoring components: a *Process Component*, a *Network Socket Component*, and a *Network Packet Component*; the Network Socket Component in conjunction with the Network Packet Component form the abstract *Network Component*. There is only a single logging component: the *Logging Component* accumulates and outputs data as required by external consumers. An overview of the general monitoring approach is presented in

Figure 3.2 on the facing page.

Process Component

The Process Component captures and evaluates information on forking, execution, and termination of Unix processes. This component uses a netlink socket in user space to subscribe to process events from the kernel [26]. This information is used to build an internal view of the process hierarchy, in which new process events can be located quickly. With the knowledge from the batch system about its processes, the subtrees for each batch job in the batch system can be identified.

For each process, primary data such as the start time, process id, parent process id, id of the user that owns the process, and process name are known, in addition to metadata used for monitoring itself. Metadata marks the network monitoring state for each process: A process is either ignored for network monitoring, a candidate for network monitoring in the future or actively monitored for its network activity. This avoids tracking network activity for processes outside of batch job process hierarchies. We propagate this metadata to child processes, simplifying the decision for new processes by using the known state of the parent process. Once a process has been excluded from monitoring, its entire subtree can be excluded as well.

Network Socket Component

The Network Socket Component links process monitoring and network monitoring. It reads the proc filesystem (procfs) filesystem to get network sockets used by each process. This approach is based on extracting used inodes, which is in principle also applicable for files. Each socket inode is mapped to the corresponding network connection, again reading procfs directly. This provides the local and remote address; the mapping to processes provides their monitoring state, that is whether the network should be monitored.

Network Packet Component

The Network Packet Component inspects a stream of network packets on all network interfaces of the host. The component uses libpcap [215], a system-independent library implementing network packet capturing APIs of pcap in the user space. For both UDP and TCP network packets, the header is copied into a buffer for inspection. Each packet's local and remote address is used to map it to a network socket, and in turn to the corresponding Unix process. The packet size corresponds to the traffic on this connection.

The design and implementation of the tool are based on C/C++. Each of the monitoring components is implemented as an individual thread. The main priority is the Network Packet Component to enable a fast processing of incoming packets.

Logging Component

Collected monitoring data is passed to the Logging Component, which performs the actual aggregation and sampling of data. Hence, it accumulates data for a given time interval and outputs a stream of results. The Logging Component supports configurable use cases and certain levels of detail. This is for example used to provide a concise overview of every batch job to the batch system.

Table 3.2.: Available process information for user-centric batch job monitoring

Name	Description
tme	Timestamp of process event
pid	Process ID
ppid	Process ID of parent process
gpid	Process ID of grouping process
uid	User ID
name	Name of a process
state	State of the process (start / exit)
error_code	error code of a process
signal	signal of a process

For this thesis, we use an output stream detail matching our requirements for workflow monitoring (see Section 3.2.2 on page 31). For each batch job managed by the batch system, two streams provide process and network monitoring data, respectively. The process stream provides process events, containing meta data of processes and encoding the hierarchy with a Dewey Index [229]. The network stream provides connection samples, providing meta data on connections held by processes in the hierarchy.

3.2.4. Data Recording at GridKa

Since July 2014 the process and network monitoring tool BPNetMon is deployed at the GridKa data and computing centre. The configuration in use can be reviewed in full in Appendix B.1 on page 157. The tool monitors the batch jobs running in the batch system, implicitly monitoring the contained pilots and payloads as well. Information on the Unix process tree of each payload and related data flows are collected. For this thesis, the monitoring stream is captured and stored persistently. This allows us to replay the monitoring stream for reproducible analysis. This forms the dataset used throughout the following Chapters for demonstrating and evaluating proposed approaches.

The data contain detailed network and process information as well as the tree structures of the Unix processes for each batch job. Each batch job is translated to a single multidimensional time series: All data is time series regarding the network traffic (traffic rates, count of inbound and outbound packets, destination and source IP as well as ports), Unix process information (pid, ppid, uid) as well as information about the batch job itself. All data points across all categories have an implicitly synchronised order given by their timestamp. Thus, each batch job is described by a stream of process events as well as a stream of traffic events. Each process contains explicit information about its parent process, encoding the hierarchy of processes. Each traffic event can unambiguously be attributed to a specific process. Tables 3.2 to 3.3 on pages 34–35 give an overview of information of the stream of process events as well as traffic events.

The stream of process information events is synchronous with the associated system events. As the network stream aggregates multiple packets to connections, an independent interval must be chosen. For the data used in this thesis, a separate record of accumulated traffic information is emitted every 20 s. On the one hand, this matches sampling intervals

Table 3.3.: Available data flow information for user-centric batch job monitoring per measuring interval

Name	Description
tme	Timestamp of measuring interval
pid	Process ID of associated process
ppid	Process ID of associated parent process
gpid	Process ID of associated grouping process
uid	User ID of associated user
in_rate	Incoming data rate in kB/s
out_rate	Outgoing data rate in kB/s
in_cnt	Number of incoming network packages
out_cnt	Number of outgoing network packages
conn_cat	Category of connection (intern / extern)
source_ip	IP address of source
dest_ip	IP address of destination
source_port	Port of source
source_dest	Port of destination

considered in the literature [3, 185]. On the other hand, storing data for reproducibility means that we must limit the total volume to available storage. We assume eight connections per batch system slot, two network interfaces with ingoing and outgoing traffic, and a traffic event size of 1 kB to include connection data, process id as well as traffic. With our chosen interval and an average runtime of 10 h per batch job (see Section 8.1.1 for detailed statistics on our present dataset), this amounts to 60 MB per batch job or 1.5 GB for all batch jobs running on a worker node at once.

A typical HEP batch job as recorded by BPNetMon is shown in Figure 3.3 on the next page. Such a batch job is composed of processes specific to the batch system, the pilot paradigm as well as the actual payloads, that is the jobs executed by users. A batch job can consist of millions of different processes, many of which have a short duration. For visualisation, all processes with a duration of less than 10 s have been excluded. Processes with related data flows are coloured relative to their traffic. This in-detail view of data flows allows recognising patterns in batch jobs.

To avoid manual analysis of collected batch system statistics and batch job data whenever an incident is suspected, we target an online anomaly detection. Based on the data collected by our tool, we presume that this is sufficient for an anomaly detection at runtime. However, as each batch job is described by its processes and their hierarchy, each monitoring event stream describes a dynamic tree. Thus, we require an online analysis for dynamic semi-structured data.

3.3. Implications for Online Analysis

The continuous streams of monitoring data generated by our distributed sensors are the basis for an online analysis of HEP batch jobs in the GridKa. However, both the methodology

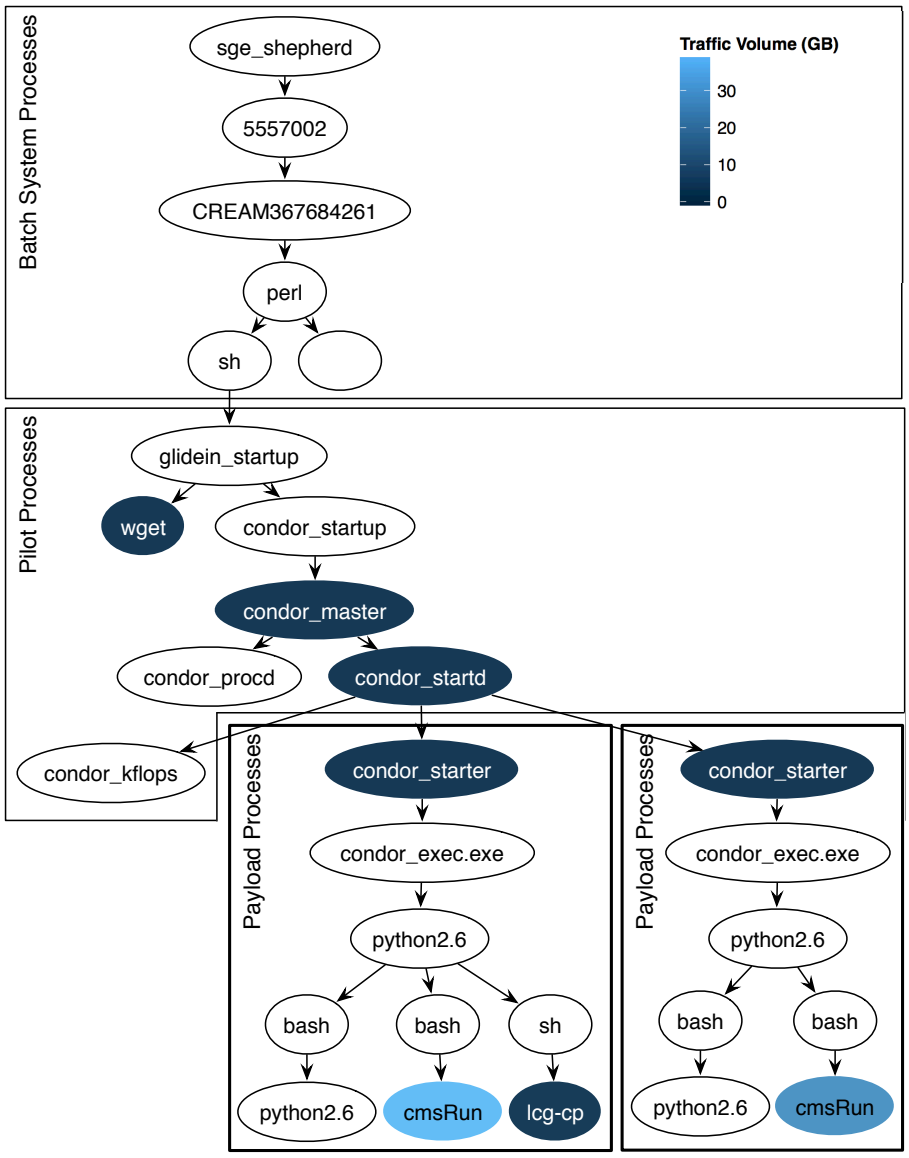


Figure 3.3.: Visualisation of a typical CMS pilot batch job. The batch job consists of processes specific to the batch system, processes from the pilot as well as processes that belong to the actual payloads. The processes dedicated to the payloads are of particular interest for user-centric monitoring. Furthermore to classify the different payloads we require the monitoring of traffic information. Processes with related network traffic are coloured depending on traffic volumes.

and the use case itself have a number of implications and challenges for online analysis. To the best of our knowledge, there is no existing framework or technique suitable for our use case. Thus, we focus on the adoption and extension of related techniques to our requirements. In the following, key aspects and challenges for online analysis of batch jobs are discussed and summarised.

The operation in a production batch system poses several implications for an online analysis of batch jobs (compare Section 2.3 on page 20). Most importantly, available resources are limited. This encompasses processing, storage, and network resources, which are dedicated to the execution of batch jobs, not their monitoring or analysis.

However, our use case dictates a high complexity of data: Monitoring at process granularity is required to distinguish pilots from payloads and isolate jobs. The differentiation of individual workflow types hinges on traffic patterns. Thus, an analysis must efficiently handle streams of dynamic tree data.

We aim for an immediate reduction of data, to reduce both the data volume and complexity of dynamic, semi-structured data. With the goal of classification and outlier detection, this lends itself to performing stream-based distance measurements. Reducing relations between trees to distances provides a robust basis for further data analysis [16, 182, 212]. Consequently, we strive for a distance measure for dynamic trees, on which distance-based analysis from literature can be applied [147].

The streaming environment adds severe constraints for tree distance measurements. Data streams imply trees of possibly infinite size; for our use case, data must at least be assumed to be considerably larger than available memory. Therefore, space must typically be restricted to logarithmic or at least sublinear complexity. Consequently, any approach must work with a compressed view of required data. Still, backtracking or explicit access to arbitrary data sections is excluded in a streaming environment. Both the data representation and distance must be computable in a single pass over the data and with small per-record processing time.

Furthermore, an appropriate measure that makes jobs well-separable in a metric space needs to be derived. On the one hand, it must be compatible with the lack of some features commonly assumed for tree distances. Most prominently is a lack of deterministic traversal order, as trees may grow and shrink concurrently at arbitrary branches. On the other hand, available information must be leveraged extensively. This includes time series attached to vertices as attributes but also derived features such as the lifetime of vertices.

Finally, the underlying use case presents some additional challenges. Whenever a workflow changes logically or technically, for example, due to an update of an underlying software, the features of jobs are also expected to change. Later analysis stages must be able to deal with such non-stationary data over long timespans. At short timespans, the processing environment implies noise. This is a direct result of the concurrent process execution order not being handled deterministically by the operating system.

These aspects mentioned above include challenges regarding resource utilisation, stream processing, handling of noise, distance measurement for dynamic trees, and online analysis itself. In the following, we give a summary of the challenges and requirements we strive to address.

Resource Utilisation A key challenge is resource utilisation. As the analysis itself is located within the WLCG site, the utilisation of processing, storage, and network resources must be minimised to not impact production operations.

Stream Processing Efficient stream processing requires one-pass algorithms for data handling with logarithmic, polylogarithmic, or sublinear space complexity at worst. Thus, approaches must focus on incremental analysis to reduce computational cost by incrementally using intermediate results.

Online Analysis An online analysis of long-running HEP batch jobs must guarantee continuous intermediate results based on a distance measure for dynamic trees. Most importantly, intermediate results may not invalidate earlier information.

Distances for Dynamic Trees Due to our choice of monitored features and granularity, an online analysis needs to deal with dynamic trees describing the batch job, including pilot and payloads. To enable non-proprietary data analysis, we focus on distance measures for dynamic trees.

Noise and Micro Changes A distinct challenge of the environment itself is the presence of noise. Several sources of noise need to be considered: noise from the operating system as well as the non-deterministic execution of concurrent processes resulting in an arbitrary order of events.

To not overcomplicate the analysis of these features, we assume that some simplifications are met – while they cannot be strictly proven, they match observations of collected data. The impact of these features on our work is low, and would mostly affect the statistical analysis.

Notably, we explicitly exclude interferences and correlation between concurrent batch jobs. Instead, we refer to the work of Kambadur et al. [127] that suggests techniques to approach scalable application interference in complex environments. Building on this, we do not consider dependency graphs to examine inter-batch job dependencies. However, it is in principle possible to include inter-batch job dependencies in the analysis of network connections. Furthermore, the recording and analysis of file accesses would allow studying intra-batch job dependencies.

4. Formalisation of Distances for Dynamic Streaming Trees

The background of this work necessitates the handling of streams describing dynamic trees. The analysis of this data is based on similarity and distance measures between such trees. In this Chapter, an overview of basic tree distance measures is given. Based on this, we introduce a generalised, modular tree similarity measure suitable for stream processing.

The tree distance approaches presented in literature are applicable for different classes of trees. In the following, we differentiate between static trees, dynamic trees as well as trees in streaming environments. Most of the current methods deal with static trees only. Only some provide a solution for trees in streaming environments or even dynamic trees.

To abstract from the underlying classes of trees, we first define a tree representation that is suitable for any class of tree. For this, we propose a concise notation and an underlying framework for the encoding of trees. Our approach is discussed based on linear tree similarity measures, illustrating key features of both our approach and the use case. These preliminaries form the foundation for later Chapters, where we present advanced tree encoding strategies suitable to deal with attributes as well as more complex special cases. Some parts of this Chapter have been published in Kuehn and Streit [138].

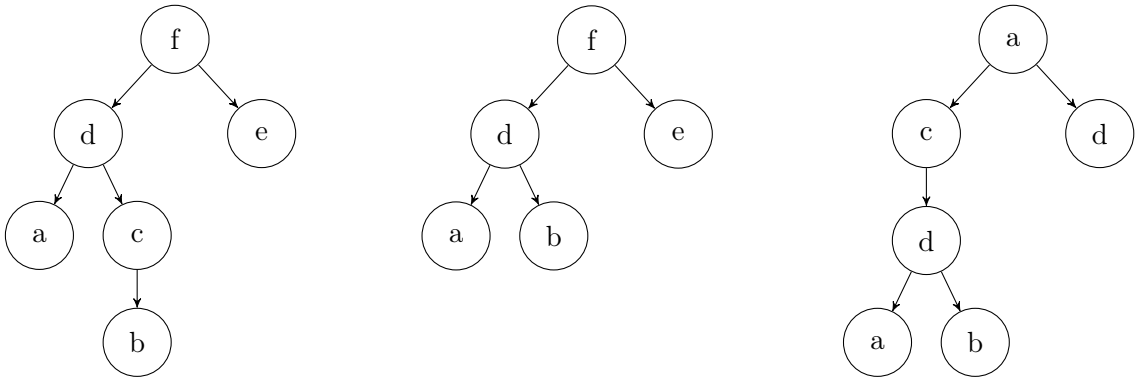
4.1. Related Work

Tree similarity or distance has been studied extensively for decades. The most important method to measure distances between trees is the Tree Edit Distance (TED). The TED is a natural generalisation of the edit distance from the domain of strings. Similar to the string edit distance [7], the TED determines the distance between two trees by the minimum number of required edit operations to convert one tree into the other [209]: the insertion, deletion, and relabelling of vertices. An example for the TED and its underlying vertex mapping is given in Figures 4.1 to 4.2 on pages 40–41. TED is still widely recognised as the state-of-the-art similarity measure for tree-structured data [150]. However, its high computational complexity limits its applicability.

4.1.1. Tree Edit Distance

Exact algorithms to compute TED are computationally expensive. The best-known method to determine distances for ordered trees has at least $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space complexity [49, 174, 178] for a number of n vertices of a given tree. While distances for ordered trees can be solved in polynomial time by utilising dynamic programming [75, 209, 239], unordered trees are NP-complete [109, 240].

However, even for ordered trees, the problem is NP-hard when the subtree move operation is introduced [163, 199]. For an exhaustive overview of TED-based approaches and its variants, such as largest common subtree [8], smallest common supertree [102], or tree alignment [121], we refer the interested reader to Bille [41] and literature referenced within.



(a) Example tree T_1. (b) Intermediate tree T_1'. (c) Example tree T_2.

Figure 4.1.: Application of tree edit operations to transform tree T_1 into tree T_2. To transform tree T_1 given in Figure 4.1a into T_2 given in Figure 4.1c deletion, insertion as well as relabelling is required. Figure 4.1b visualises the intermediate step after deletion of vertex c.

Many use cases such as change detection [57], fraud detection [231], information visualisation [11], or our use case introduced in Chapter 3 on page 23 require approaches that are scalable and feasible in almost near real-time. Especially in streaming environments, it is not feasible to rely on polynomial time algorithms. Thus, we further concentrate on approximate algorithms for TED.

4.1.2. Approximating Tree Distances

There are many efforts in developing approximate algorithms for TED. While TED-based algorithms consider the original trees for distance calculation, approximate algorithms usually rely on simplified representations of trees to reduce computational complexity for distance calculation. Approximation methods for TED usually target different features for representation, such as structure, content, or both structure and content combined.

In the following, we distinguish three classes of tree distance approximation methods: summary-based, decomposition-based, and time series-based approximation. We provide a short introduction to each of the three classes of tree distance approximation. We also compare the three classes with regard to the constraints of a streaming environment as well as dynamic trees.

Summary-Based Approximation

Summary-based approximation methods represent a tree by summarising selected features. Usually, summary-based methods represent trees as vectors or matrices of numerical values or hashes [59, 125, 149, 221, 223, 237] to optimise space and time complexity.

Cruz et al. [72] model trees based on distribution analysis of vertex frequency measures. This compares independent statistics on the repetition of structural features in each tree. However, the underlying vertex model does not consider the relative order of vertices, and the statistical approach offers limited sensitivity for small differences. Thus, it is adequate only when data is drastically different from each other.

Chowdhury et al. [59] filter significant properties of a tree based on results from information

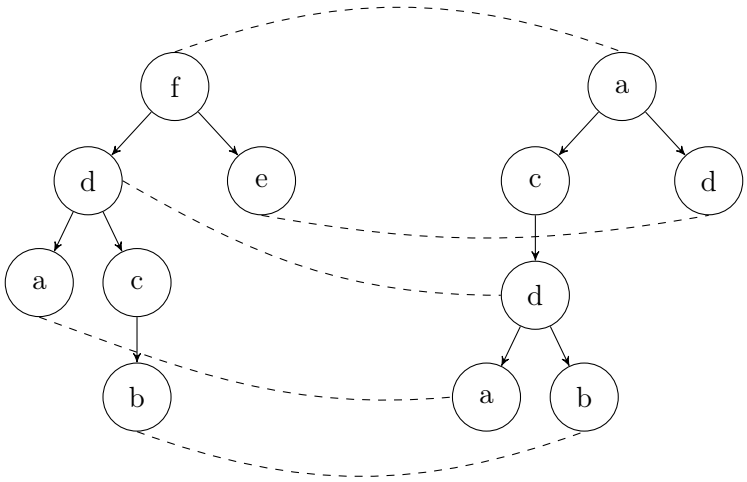


Figure 4.2.: Node mapping corresponding to Figure 4.1. All vertices that are mapped by the dashed lines and do not share the same label, are renamed. All vertices that are not mapped in tree T_1 are deleted whereas all vertices that are not mapped in tree T_2 are inserted to T_1. The Tree Edit Distance is a measure for the minimal cost required to map the vertices of T_1 to T_2. In this example, the TED is 4. That means, each operation has the same cost of 1.

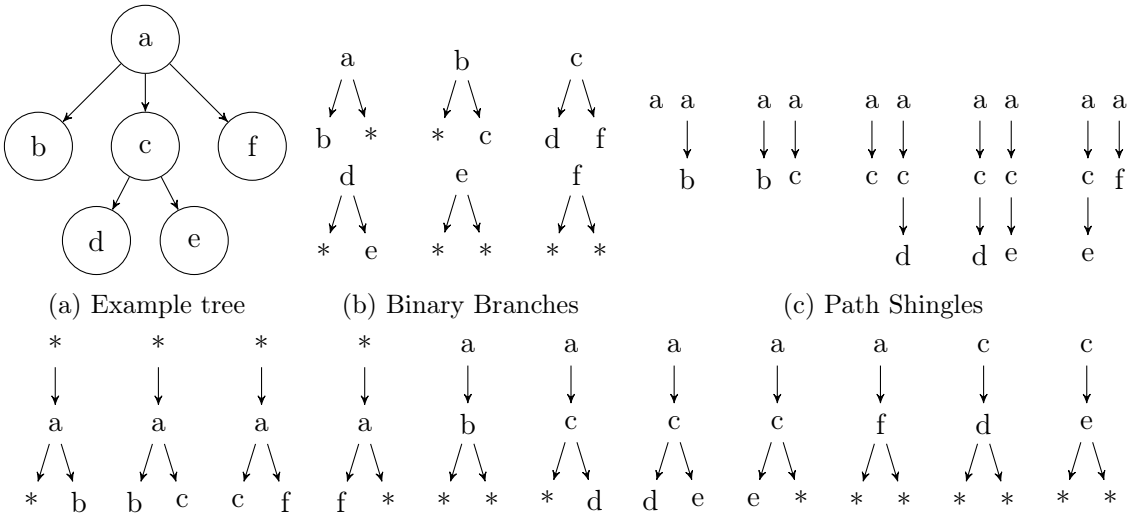
retrieval, showing that the most and least frequent properties often add no semantic meaning [159]. Thus, their approach removes the most frequent and infrequent properties to obtain a reduced tree representation for distance approximation. However, this approach requires offline analysis of tree statistics to construct the summary. This is not feasible for possibly infinite streams of trees, or to derive results before the end of the stream.

Lian et al. [151] summarise a set of trees by a directed graph which consists of the set of vertices and edges appearing in either tree. Hence, this graph encodes the structure over the entire given set of trees and is called the s-graph. Distance calculation for two s-graphs is based on the number of edges that are not shared by the given s-graphs. The proposed method targets clustering of trees. Thus, the definition of s-graphs is based on summarising sets of trees. A distance calculation for two individual trees is a special case as each s-graph representation is the actual tree itself. Comparisons are performed by dividing trees into sets of individual edges. Consequently, from our point of view, the proposed method can also be classified as a decomposition-based approximation. However, the method requires for n vertices in a given tree $\mathcal{O}(\alpha n)$ space where α is a small constant factor and $\mathcal{O}(n^2)$ time complexity. Especially the time complexity renders this approach infeasible for our use case.

Decomposition-Based Approximation

A tree decomposition is an alternative to edit-based distances. By utilising tree decomposition, similarity or distance between two trees is calculated based on a set, multiset, or sequence of substructures of the tree, so-called *snippets*[1] [22, 25, 49, 70, 94, 120, 151, 214,

[1]Depending on the field of research different terms are used for the same concept to express the decomposition into smaller pieces: common terms are shingles, snippets, tokens, or twig patterns. Within this thesis, we adopt the term *snippet* for uniformity.



(a) Example tree (b) Binary Branches (c) Path Shingles

(d) pq-grams

Figure 4.3.: Overview of selected tree decomposition methods that are used for tree distance approximation. Figure 4.3a visualises an example tree that is the base for the three different decomposition methods. Figure 4.3b shows the resulting representation based on decomposition by the concept of *binary branches*. Each subtree pattern is comprised from an anchor vertex, its left-most child as well as its right sibling. Figure 4.3c shows the approach of path shingles. For a window size of 2 the different partial paths from a specific vertex to the root vertex are determined. In Figure 4.3d the approach of pq-grams is visualised. A pq-gram describes a tree pattern from $p-1$ parent vertices as well as q child vertices. The example visualises a configuration of $p = 2$ and $q = 2$.

233] to represent a tree. Intuitively, two trees are similar if their decompositions have many snippets in common.

Proposed methods from literature usually differ in the type of snippet they consider for decomposition. Common snippet types are sub-trees, graphs, paths, edges, individual vertices as well as vertex contents and attributes (see Section 4.3 on page 46 for an overview of naming conventions for trees). An overview of selected decomposition methods is given in Figure 4.3.

Some of the methods we discuss in this Section, including binary branches, pivot structures, or valid subtrees, have been originally proposed to be represented directly as vectors. As those methods are algorithmically comparable to decomposition, we introduce their underlying decomposition method individually for completeness. In contrast, the inverse is the case for decomposition methods. Decomposition results can be embedded into a vector space if required, for example, to realise a compact representation for implementation.

In general, approaches focusing on the decomposition of tree- or graph-shaped patterns can retain more complex structural properties. As a result, several of these methods efficiently approximate TED [22, 94, 233] while providing better scalability.

Decomposition to Primitives The simplest decomposition approaches use primitive elements of trees: vertices, edges and paths. While these elements retain only a minimum of structural information, they allow for compact representations.

Individual vertices are the most primitive elements of a tree, retaining no structural information. The most fundamental hierarchical relationship for a vertex is its edge to its parent. Consequently, a path is composed of edges; each path is a list of consecutive nodes in a tree, based on recursive parent-child relationships. Approaches differentiate between root to leaf paths as well as partial paths. A partial path is a path from any vertex within a tree up to the root vertex.

While a decomposition utilising paths, edges, and/or vertices mainly represents structural properties of a tree [73, 123, 151, 167, 211], the utilisation of vertex contents and attributes mainly focuses on content independent from any tree structure. However, the utilisation of a combination of both characteristics is considered for many use cases to increase the precision of representation and thus distance approximation [10, 166, 181, 201, 234, 236]. Other approaches, such as path shingles proposed by Buttler [49] consider a window of tokens for tree representation.

Binary Branches Yang, Kalnis, and Tung [233] propose the decomposition of a tree into binary branches. The method relies on the concept of a left-child right-sibling binary tree [66]. This concept describes how to transform any tree to a binary tree. This is based on defining binary branches for every vertex: A binary branch is a snippet that consists of an anchor vertex, its first child vertex and its right sibling. The multiset of binary branches is utilised to calculate the distance between trees based on set intersection. Binary branches provide a space complexity of $\mathcal{O}(n+m)$ and a time complexity of $\mathcal{O}(\max(n,m))$, where n is the number of vertices in one tree and m the number of vertices in the other.

Especially the space complexity does not meet the requirements of a streaming model. In addition, also the definition of binary branches itself is not feasible for dynamic streaming trees. In many traversal orders, both children and right siblings are known only well after a vertex, and the lack of either is only well known at the end of a stream.

Pivot Structures Tatikonda and Parthasarathy [214] introduce wedge-shaped snippets, so-called pivot structures. Each pivot structure consists of two vertices as well as their least common ancestor (lca) within the tree. Hence, a tree is represented by its multiset of pivot structures, which is the basis for distance approximation. The lca is not necessarily the parent vertex of the two vertices. Thus, the proposed approach does not rely on usual snippets but summarises the path between the vertices and the lca by an arbitrary edge, which is not necessarily part of a given tree.

The proposed method explicitly supports the incremental generation of the pivot representation. However, it features time and space complexity of $\mathcal{O}(n^2)$ as for each vertex within a tree, the pivot structure to every other vertex is generated. This is infeasible in streaming environments for two reasons: First, the whole tree needs to be stored to ensure pivot structure generation. Second, the time and space complexity of the approach does not scale to large trees.

pq-grams Augsten, Böhlen, and Gamper [21, 22] propose pq-grams to approximate the distance between ordered labelled trees. Following this approach, a tree is represented by all its subtrees that match a particular, predefined shape. This predefined shape consists of an anchor vertex with $p-1$ vertices on the path to the root vertex and q children. These subtrees are stored in a multiset, which is again used for distance calculations via set intersections.

To derive distances for unordered trees, windowed pq-grams [23, 25] were introduced. This method introduces an additional sorting of vertices on each group of siblings, which makes it invariant to permutations of siblings.

The pq-gram approach features a time complexity of $\mathcal{O}(n \log n)$ and space complexity of $\mathcal{O}(n)$. Furthermore, the multiset of pq-grams can be maintained incrementally [20, 24].

However, both space complexity and specifics of the subtree pattern itself render pq-grams inappropriate for streaming environments and dynamic trees. On the one hand, at least sublinear, ideally logarithmic or poly-logarithmic space complexity is required in a streaming setting. On the other hand, construction of pq-grams requires to include vertices that are not well-defined in dynamic trees. For example, when a vertex is added to a tree, its child vertices are still unknown. Consequently, pq-grams as originally proposed by the authors are not directly applicable in streaming environments.

Valid Subtrees Garofalakis and Kumar [93, 94] propose a tree decomposition into valid subtrees for ordered, labelled trees in the streaming model. The approach relies on incremental deterministic collapsing of vertices to build a compact synopsis of massive, streaming trees. This synopsis requires only small space of $\mathcal{O}(d \log^2 n \log^* n)$ and $\mathcal{O}(\log d \log^2 n (\log^* n)^2)$ time per vertex that is added in pre-order, where d denotes the depth of the tree. The synopsis can be used for approximate distance computations with guaranteed error bounds.

Although space and time complexity render the approach feasible to approximate distances for streaming trees, it relies on pre-order processing of vertices. By assuming pre-order processing of vertices, the collapsing of vertices and thus the creation of compact synopsis becomes possible. The authors only require preserving a local influence region of $\mathcal{O}(d)$ vertices to attach further vertices. However, we need to consider dynamic trees that are characterised by the fact that vertices can be added at any branch in arbitrary order. Consequently, this approach to tree decomposition for distance approximation is not feasible for dynamic trees in streaming environments.

Time Series-Based Approximation

Flesca et al. [88, 89] introduce a technique based on Discrete Fourier Transformation (DFT) to compute the similarity between trees. The basic idea is to remove all contents from the vertices of a tree, but retain the structure and the vertex' labels. Each vertex within the tree in pre-order (depth-first and left-to-right order) is represented as a number. This sequence of numbers is a time series that is converted to a set of frequencies by utilising DFT. The distance between two trees is computed by taking the difference of the magnitudes of the corresponding frequencies of the two signals.

While this approach directly supports the processing of trees in a streaming fashion by design, it still has some severe drawbacks. First, time complexity for the representation of trees as a time series is $\mathcal{O}(n^2)$ due to an application of DFT. Second, the approach removes all content-related data. Thus, attributes that are considered important for our use case are not supported at all. Therefore, an explicit mapping of structure, as well as content transformation to integer values, would be required, complicating the analysis and increasing space complexity. Furthermore, the approach relies on a constant sampling period, while we face variable sample periods for process events as well as traffic events. Most importantly, Fourier transformation typically operates on repeating, infinite time series. Consequently, the stream of a dynamic tree needs to be amplified by zero-padding data to match the size

of the second tree to be compared to. This severely impacts computational costs as sizes of streams are not known in advance.

4.1.3. Summary

In general, one can say, that each of the stated approximation classes may reduce structural or content-based information to some extent. Each of those methods calculates some representation for the original tree with the purpose to reduce computational complexity while retaining accurate distance values.

Summary-based approximations usually have superior space complexity compared to the other classes of tree distance approximation. It is even possible to get constant complexity. However, regarding utilisation in streaming environments, summary-based methods have several disadvantages. The method itself involves the task of feature selection, for example determining which statistics are most relevant to represent domain-specific trees. Possibilities include statistics of structural properties such as fanout, depth, number of leaves or content-based properties such as relative frequency of labels or attributes. However, determining such statistics usually requires an offline learning process. Thus, distance approximation in stream environments is either delayed, or intermediate results have a small statistical relevance.

Decomposition-based approaches reflect interesting properties such as the incremental creation of representation while providing appropriate space and time complexity. However, none of the available methods is directly applicable to dynamic trees in streaming environments. Still, the general concept of utilising snippets such as trees, graphs, paths, edges, or vertices can be considered.

Finally, the time series based approximation has some very promising properties, such as independence from a specific traversal order of vertices. This can be regarded as one of the key requirements of dynamic trees. However, computational complexity is not suitable for a streaming model. Furthermore, the utilisation of Fourier transformation causes difficulties regarding different size of streams as well as differing intervals for traffic and process events present in our data.

4.2. Overview of the Approach

Analysing static or dynamic trees in streaming environments requires a simple, lightweight representation to overcome the curse of dimensionality [56] and enable online analysis. Both classes of trees can be modelled as a sequence of gradually differing fixed states. Thus, an incremental representation of trees is desirable. In the context of tree distances, this facilitates a similarly dynamic measure based on differences between states. As repeated calculation of tree distances for every fixed state is inefficient, the distance measure is also desired to work incrementally.

Reflecting this, this thesis presents a method using two distinct tasks to enable online analyses for dynamic trees:

1. A *gradual tree decomposition* to represent trees via independent elements, suitable for streaming (see Section 4.4 on page 49).

2. An *incremental tree distance measure* to build on this representation, converging towards classical tree distances (see Section 4.5 on page 67).

The separation into two tasks, namely a proper tree representation and a distance defined on it, simplifies the formalisation and extension of the approach. Tree decomposition is widely used in literature to facilitate embedding trees into simpler data structures, such as vectors or sets [59, 100]. Basing the approach on decomposition makes it comparable with embedding-based distance measures from literature. Separating representation and distance allows switching to differing approaches as well easily.

Furthermore, each of the two methods allows for different optimisations. Decomposing a tree to a simpler representation focuses on reducing dimensionality, memory, and processing requirements while retaining significant features. In contrast, the distance function aims at maximising the information differentiating trees. Thus, differentiating these methods allows optimising for specific use cases.

This modular approach is not restricted to the two-step procedure of tree decomposition and distance measurement. Further methods can be added to extend proposed functionality. For example, the tree decomposition can be used as a basis for probabilistic sketching [5, 67, 103, 112]. This allows for continued flexibility of the overall approach beyond the use case described in this thesis.

Graph-based exact distance measures are computation- and memory-intensive. The well-established TED has a complexity of $\mathcal{O}(n^2)$, making it unsuitable for stream processing. Embedding-based methods attempt to remedy this by projecting selected features into a low-dimensional, often Euclidian space.

Specifically for the embedding of trees, different approaches are available in literature. Each method summarises different statistics of the trees under consideration to encode identifying features [59]. For example, the appearances of vertex labels are analysed by frequency, and the ten most frequent vertex labels are used for the tree embedding. Other methods are based on tree decomposition, subsequently dividing a tree into smaller parts.

The statistical approaches are not feasible in streaming environments. An embedding based on statistical analysis requires a repeated update or even replacement of the current tree embedding to account for changes. Therefore, the complexity of distance calculation is driven by the number of required changes m and the respective size of the tree n, resulting in a complexity of $\mathcal{O}(mn)$.

Instead, approaches based on tree decomposition allow for an incremental update of the embedding by focusing on the current changes that are inserted into the tree. Thus, an incremental distance measurement can be applied to the current change only, enabling a complexity of $\mathcal{O}(m)$.

Based on this discussion, we propose a two-step procedure that distinguishes between the decomposition-based tree embedding and the actual distance measure. While this approach is suitable for many use cases, we specifically focus on incremental processing. In specific, we establish the approach to process dynamic trees in streaming environments.

Figure 4.4 on the facing page gives an overview of the two-step procedure enabling online analysis of dynamic trees in streaming environments. A tree is decomposed to build an embedding which approximates relevant features. The distance measurement between trees is independent of the original trees and only relies on the existing tree embeddings.

4.3. Preliminaries

The work presented in this thesis is focused on labelled, ordered trees. This directly reflects the use case of describing batch jobs by their process hierarchy. However, unlabelled or

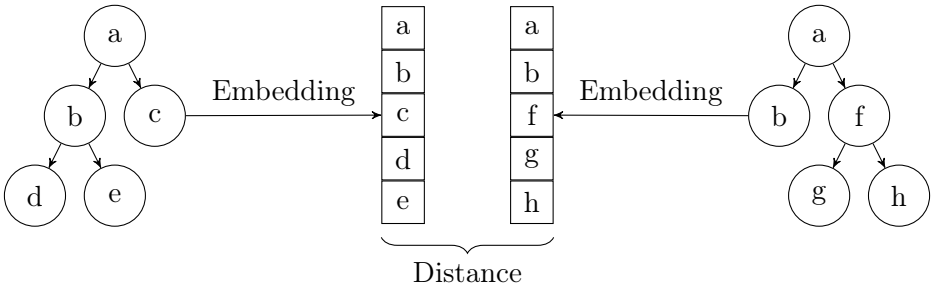


Figure 4.4.: Visualisation of the two-step procedure to approximate distance measurement for dynamic streaming trees. The visualised approach distinguishes the decomposition of any given tree to an approximate embedding and the distance measurement based on the given embeddings. While the decomposition creates an approximate embedding, the distance measurement calculates exact results.

unordered trees can be trivially converted via arbitrary labelling and ordering. The most commonly used terms for trees are denoted in Figure 4.5 on the next page in relationship to a specific vertex. To describe the hierarchical relations between vertices of a tree, we use a notation commonly used in literature.

4.3.1. Basic Notation

A rooted tree $T = (V, E)$ consists of a finite set of vertices or nodes $V(T)$ and a finite set of edges $E(T)$. The size of T, meaning the number of vertices in T, $|\,\mathrm{V}(T)|$, is denoted by $|T|$.

Each vertex $v \in \mathrm{V}(T)$ has zero or more child vertices. A vertex is a *leaf vertex* if it has no children and an *internal vertex* otherwise. A vertex that has a child is called the child's *parent* vertex and we denote the parent of vertex v by $v.\,\mathrm{parent}$. The topmost vertex that has no parent is the *root* of T and is denoted by $T.root$.

The *fanout* of a vertex $v \in \mathrm{V}(T)$, denoted as $v.deg$, is the number of children of v. The *depth* of a vertex $v \in \mathrm{V}(T)$, denoted as $v.depth$, is the number of edges on the path from v to $T.root$. In specific, the depth of the root vertex $T.root.depth$ is 0.

Let $\mathrm{T}(v)$ denote the subtree of T rooted at a vertex $v \in \mathrm{V}(T)$. All vertices $w \in \mathrm{V}(\mathrm{T}(v)) \setminus \{v\}$ are *descendants* of v. For $w \in \mathrm{V}(\mathrm{T}(v))$, v is an *ancestor* of w. In other words, v lies on the path from w to $T.root$.

Two vertices $u, v \in \mathrm{V}(T)$ are *siblings* if they have the same parent, that is $u.parent = v.parent$. A tree T is ordered if any left-to-right order among the siblings is given. For an ordered tree T with root v and children $v_1, \ldots, v_i$, the *level-order* of $\mathrm{T}(v)$ is obtained by visiting v and then visiting $v_1, \ldots, v_i$. A list of vertices $u \in \mathrm{V}(T)$ is created by adding all children of visited vertices in order. The vertices in u are visited in order while still appending children to the list.

A list of vertices preceding a vertex v in level-order is obtained by $v.\,\mathrm{level}$. The level-order number of a vertex $v \in \mathrm{V}(T)$ is the number of vertices preceding v in the level-order traversal of T, and is given by $|v.\,\mathrm{level}\,|$. To refer to the position of w among the children of $\mathrm{T}(w.\,\mathrm{parent})$, we define $w.pos$ for convenience, that is the level-order number of $w \in \mathrm{V}(\mathrm{T}(w.\,\mathrm{parent}))$. The vertices to the left of $w \in \mathrm{V}(T)$ form the list of vertices $u \in \mathrm{V}(\mathrm{T}(w.\,\mathrm{parent}))$ such that $u.pos(u) < w.pos$. Furthermore, we assume total ordering: if u is to the left of w then w is to the right of u.

We assume throughout this thesis that labels assigned to vertices, denoted by $v.\,\mathrm{lbl}$, are

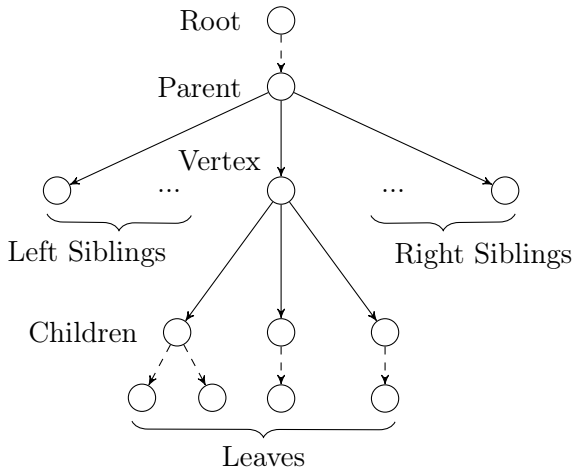


Figure 4.5.: Components of a tree in relation to a given vertex. The descendants of the vertex are called its children. All vertices that have no children are called leaf vertices. The direct ancestor of the vertex is its parent. The vertex without a parent is called the root vertex. The vertices to the left und right of a given vertex are called siblings.

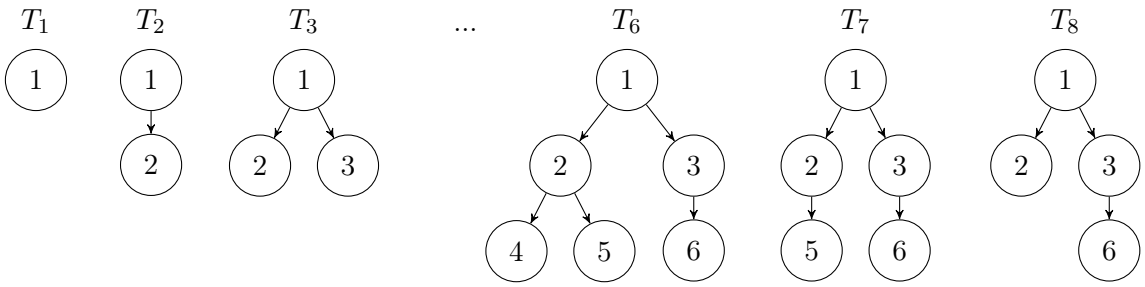


Figure 4.6.: A dynamic tree $\mathscr{T}$ with is sequence of snapshots $T_1, \ldots, T_8$. The labelled vertices represent the same vertex in each snapshot $T_i \in \mathscr{T}$.

chosen from a finite alphabet Σ. Furthermore, let $\lambda \notin \Sigma$ denote a special blank symbol. and define $\Sigma_\lambda = \Sigma \cup \lambda$.

4.3.2. Dynamic Trees

Entities represented via trees are not necessarily static, requiring the trees to change their structure. This change includes the insertion or removal of vertices or the modification of a vertex's attribute. We call a tree which evolves over time a *dynamic tree*. The terms temporal tree or evolving tree are commonly used in literature as well. For better distinction, we denote a tree that does not change as *static tree*.

Definition 4.1 (Dynamic Tree)**.** A *dynamic tree* $\mathscr{T} = (T_1, \ldots, T_n)$ is a sequence of static trees $T_i = (V_i, E_i)$, with every T_i representing the state of $\mathscr{T}$ at a given moment i.

For a given dynamic tree $\mathscr{T} = (T_1, \ldots, T_n)$ such as the one in Figure 4.6 we call each single tree in the sequence a *snapshot* of $\mathscr{T}$ and write $|\mathscr{T}|$ to denote the number of snapshots in $\mathscr{T}$. The difference between two snapshots can be expressed by a number of edit operations. These edit operations transform one snapshot into the other.

In this thesis, we differentiate three atomic edit operations: the insertion, removal, and modification of objects. Without loss of generality, we limit the set of possible objects onto which an edit operation can be applied on, to the set of vertices of a tree T, that is $\mathrm{V}(T) \cup \varnothing$. Thus, we differentiate three atomic edit operations that only affect vertices and their attributes:

1. the insertion of a new vertex v: $\varnothing \to v$,
2. the removal of a vertex v: $v \to \varnothing$, and
3. the modification of an attribute a_n of a vertex v: $a_n \to \tilde{a}_n$,

where $\varnothing$ denotes an empty vertex. However, the same operations are in principle applicable to edges as well.

Other approaches in the context of edit distances, both for string [68, 210] and tree edit distances [94], also include move operations. This additional operation relaxes the underlying problem of aligning objects with each other. However, those methods still face the challenge of dealing with nontrivial alignments. An explicit move operation is excluded in this work but corresponds to the removal and insertion of the specific vertex.

Any two snapshots can be related to a finite number of edit operations. Throughout this thesis, we assume the sequence of snapshots to be complete: any snapshot T_i differs from T_{i-1} and T_{i+1} by exactly one edit operation. This creates a symmetry between dynamic trees and tree streams: The sequence of snapshots of a dynamic tree corresponds to the stream of operations building a tree.

4.4. Decomposition-Based Tree Embeddings

Incrementally processing trees in streaming environments requires us not to consider the entire tree. This lends itself to the usage of decomposition methods: instead of operating on the entire tree, the tree is decomposed into smaller entities of which only few must be processed at any time. Given our use case, we decompose trees both structurally and temporally into the currently active elements at the granularity of individual vertices.

Decomposing trees allows the use of embedding techniques: elements from the tree are *embedded* into a simpler data structure. The semantics of translating a position in the tree to the embedding are tuned to preserve relevant information of the tree structure. At the same time, the simplicity of the data structure itself allows for more efficient operations.

In this Section, we take an abstract approach to embedding. The focus is not on any implementation, but a formalisation suitable to select specific techniques as required by individual use cases.

4.4.1. Vertex Identities

Tree decomposition must target the lowest common denominator of trees to make it meaningful and comparable: atomic edit operations. This is adequate for both dynamic and static trees, with the latter being restricted to creation of vertices. To compare two edit operations, we compare their type and subject, namely the *vertex* being *edited*. While the type of edit operation is trivial to compare, the vertex is not.

Comparing vertices requires knowledge about each vertex' defining features and location in the tree. With the goal of vertex-based decomposition, we restrict the description of the

latter to a limited neighbourhood of the vertex. In general, we call the defining features and neighbourhood, or context, of the vertex the vertex' *identity*.

Definition 4.2 (Identity)**.** The *identity* $\mathrm{Id}(v)$ of a vertex $v \in \mathrm{V}(T)$ encodes a collection of characteristics of the vertex' context within the tree T as well as significant attributes of v.

Example 4.3 (Identity)**.** Suppose the sequence of relevant features are a vertex' label as well as the labels of its parent, and its left and right sibling. Consider a root vertex a labelled a with children b, c, d, e, and f labelled likewise. Exemplarily for the two vertices a and d, this results in the identities $\mathrm{Id}(a) = (a, \varnothing, \varnothing, \varnothing)$ and $\mathrm{Id}(d) = (d, a, c, e)$.

Property 4.4 (Finite Number of Identities)**.** Let the label of a vertex $v \in \mathrm{V}(T)$ be the defining feature of its identity $\mathrm{Id}(v)$. Based on the assumption that labels are chosen from a finite alphabet Σ_λ (compare Section 4.3.1 on page 47) and a finite number of attributes to encode it follows that the number of distinct identities $A = \{\mathrm{Id}(u) \,|\, u \in \mathrm{V}(T)\}$ with $|A|$ is also finite.

The practical decision of what constitutes the relevant features and context of a vertex strongly depends on the use case. It is worth pointing out that the identity of a vertex is not necessarily unique in a tree. Since the context is limited to a subset of the tree, multiple vertices within the same tree can result in the same identity.

Further discussions are largely independent of the actual choice of an identity. Thus, to represent an *abstract identity*, we borrow the bra-ket notation [77] used in physics. We adopt this notation for brevity, without attempting to replicate the complete underlying mathematical rule set. For any given vertex v, we define its identity Id as $\mathrm{Id}(v) = |v\rangle$. Furthermore, we define the similarity of two vertices v, w based on their identity as their *projection* $\langle v|w\rangle$. Based on the concept of an identity for the vertices of a tree, we derive a representation of an entire tree T by its collection of identities.

4.4.2. Identity Profiles for Trees

A tree T is composed of its set of edges $\mathrm{E}(T)$ and vertices $\mathrm{V}(T)$. Thus, from the given projection of vertex identities, we can derive a representation of an entire tree T. This representation acts as the *identity profile* of the tree.

Definition 4.5 (Identity Profile)**.** The *identity profile* $\mathrm{Id}(T)$ of a tree T is a collection $(\{\ldots\})$ of the tree vertex identities, i.e. $\mathrm{Id}(T) = (\{\mathrm{Id}(v) \,|\, v \in \mathrm{V}(T)\})$.

Although the identity profile does not contain explicit information on edges of the tree, it implicitly covers relevant information by including the vertex' context. Depending on the choice of context, for example, coverage of child and parent vertices, the vertex hierarchy defined by edges is covered appropriately.

Reflecting the notation for vertices, we introduce an *abstract identity profile*

$$\mathrm{Id}(T) = |T\rangle = |v\rangle_{v \in \mathrm{V}(T)}.$$

This representation of identity profiles relies on the following observation.

Property 4.6. Let T be a tree with an associated identity profile $|T\rangle$. If a new leaf vertex u is attached to a vertex $v \in \mathrm{V}(T)$ resulting in a new tree T' then $|T'\rangle = |T\rangle + |v\rangle$.

The Property 4.6 on the facing page states that existing identity profiles are *not* altered when new leaf vertices are added to a tree. This is because a new leaf vertex can never change existing ancestor-descendant dependency in the tree. It enables us to build identity profiles and thus naturally support dynamic trees incrementally.

Given a dynamic tree $\mathscr{T} = (T_0, T_1, \ldots, T_n)$ with $\mathrm{V}(T_0) = \varnothing$ we define the identity profile of a sequence of vertex identities given by the sequence of n edit operations, namely

$$\mathrm{Id}(\mathscr{T}) = |\mathscr{T}\rangle = |\mathrm{V}(T_i) \setminus \mathrm{V}(T_{i-1})\rangle_{T_i \in \mathscr{T}, \text{ with } i \geq 1}.$$

The application of identities and identity profiles to streaming or dynamic environments adds requirements and restrictions that need to be considered.

4.4.3. Identities and Identity Profiles in Streaming Environments

Performing a tree decomposition in streaming environments to create identities and identity profiles raises several challenges: a) While dynamic trees naturally translate to streaming environments, static trees explicitly need to be serialised; b) the context of a vertex accessible to define its identity is restricted by the stream order as well as memory requirements; c) the independence of single decomposition operations is required to provide identities in a stream for related steps of a multi-step procedure. In the following, these requirements are addressed in detail.

Independence of Tree Decomposition Operations

To support a multi-step procedure that enables online analysis, each step needs to provide fast turnaround times while minimising information overhead for the next steps. Thus, we define the input and output for each step to be a stream or minimised stream of events. Each of these events must be independent of distant preceding and succeeding events. Ideally, each step of the procedure relies on a single input event only. This enables compact utilisation of memory of $\mathcal{O}(1)$ and guarantees single-pass algorithms. However, relaxing this to fixed length buffer or sketch of events still provides adequate memory complexity.

In turn, this means that individual events generated during tree decomposition remain independent from other events outside of a small window. Thus, individual identities must be built only from the content of an event itself, and few neighbouring events. Furthermore, stream processing allows only for look behind but not look ahead of events – future events may be delayed arbitrarily.

Tree Event Stream Representation

Processing dynamic trees in streaming environments follows naturally from basing the decomposition on the sequence of edit operations and their related vertices (compare Section 4.4.2 on the preceding page). By contrast, the stream processing of static trees is not defined unambiguously; the tree can be processed as a whole, or in slices of any sequence of its vertices. To preserve the analogy to dynamic trees, we represent static trees as a sequence of atomic edit operations on vertices.

To define the sequence of edit operations for static trees, we consider common tree traversal methods. Traversal methods can be differentiated into depth-first traversal implementing in-order, pre-order, or post-order approaches, breadth-first or level-order traversal, and various hybrid traversal methods [145]. Tree decomposition-based approaches

in literature mainly utilise pre-order traversal [17, 18, 94, 156]. While pre-order traversal is used abundantly in literature, the support of arbitrary order is to the best of our knowledge only considered by Flesca et al. [89]. Furthermore, underlying algorithms strictly require the given order of a specific traversal method, as distance results are undefined otherwise.

To accompany this issue and equally handle both dynamic and static trees, we consider any tree as an arbitrary sequence of its vertices being inserted, removed, or changed. This implies support of any tree representable as a stream of events that arrives sequentially. As a result, the formalisation presented here allows to abstract from classes, types as well as most traversal methods of trees. We define the *tree event stream representation* on any tree T with a specific traversal order of vertices σ:

Definition 4.7 (Tree event stream representation)**.** The *tree event stream representation* $S(T)$ is a sequence of n tree events $(e_1, \ldots, e_n)$ given by the traversal order σon the set of vertices $\mathrm{V}(T)$, that is

$$S(T) = (e)_{v \in \sigma(\mathrm{V}(T))},$$

where e_i is a collection describing an edit operation and the vertex subject to it.

It is important to point out that σ is restricted by consistency of the tree. For example, it is not possible to add a vertex before its parent vertex is inserted into the tree event stream representation. This implicitly excludes orderings incompatible with specific trees, such as post-order traversal for trees of infinite depth.

Each tree event e_i provides at least the type of event, the subject vertex as well as a reference to the subject's parent to deduce the position in the tree. Besides this structural information required for tree reconstruction, each tree event may contain further information depending on each use case. For example, an attribute value of a tree event can contain information on the timestamp when the event took place. Although each tree event is guaranteed to be independent, the information of the event stream as a whole is sufficient to reconstruct the original tree T from $S(T)$.

Definition 4.8 (Tree Event)**.** The *tree event* $e_i \in S(T)$ is a tuple that holds the following information:

1. the *type* of edit operation,
2. a *description* that maps the event explicitly to a distinct object and position in tree T, and
3. an arbitrary number of value attributes a.

Thus, a tree event e_i is given by $(\mathit{type}, \mathit{description}, a_1, \ldots, a_n)$.

Representing Static Trees as Insertion-Only Trees As a simplification, static trees can be interpreted as the insertion of all vertices. Thus, the number of atomic edit operations can be reduced to one, leaving the *insertion* of a vertex as the only valid edit operation. For each vertex $v \in \mathrm{V}(T)$, a single event e is appended to the event stream.

Definition 4.9 (Simple tree event stream representation)**.** The *simple tree event stream representation* $S^{\mathrm{simple}}(T)$ is a sequence of tree events that extends the definition of tree event stream representation and defines the set of possible events as $e_i \in \{e^{\mathrm{start}}\}$. For the simple tree event stream representation S^{simple} the following condition holds:

1. $e_i^{\text{start}}.node = e_j^{\text{start}}.node.parent \Rightarrow i < j$.

This definition ensures that each vertex $v \in \mathrm{V}(T)$ is present before the vertices of its subtree $u \in V(T(v)) \setminus \{v\}$ are attached to it. However, it does not guarantee that children follow directly after their parent or their siblings. The tree may grow concurrently at arbitrary branches.

Example 4.10. A tree T contains a number of vertices $v \in \mathrm{V}(T)$, where each vertex includes a timestamp t. An arbitrary order based on the timestamp t can be used to determine the event stream representation $S^{\text{simple}}(T)$. Therefore, the timestamp t is used as the order of $S^{\text{simple}}(T)$

$$S^{\text{simple}}(T) = (e = v)_{v \in \sigma(\mathrm{V}(T))}, e_{n+1}.t \geq e_n.t.$$

Representing Dynamic Trees Dynamic trees are not only created via insertion but can also change over time. New vertices may be inserted, and existing vertices may be removed at any time from each branch. In addition, values of attributes of existing vertices may be changed at any time. Therefore, the tree event stream representation for dynamic trees supports all three edit operations as events: a) the insertion of a vertex e^{start}, b) the removal of a vertex e^{end}, and c) the change of an attribute value of a vertex e^{attrib}. A pair of start event $e^{\text{start}}(v)$ and end event $e^{\text{end}}(v)$ for a vertex v marks its lifetime.

While the order between unconnected vertex events is still arbitrary, the order σ must fulfil additional constraints for consistency to be guaranteed. Specifically, vertices may only be inserted after their left siblings and after their parents. While siblings can be removed independently, all child vertices must be removed before their parent. Also, only attributes of currently existing vertices may change.

Definition 4.11 (Dynamic Tree Event Stream Representation)**.** The *dynamic tree event stream representation* $S^{\text{dynamic}}(T)$ is a sequence of events that extends the definition of simple tree event stream representation and defines the set of possible events as $e_i \in \{e^{\text{start}}, e^{\text{end}}, e^{\text{attrib}}\}$. For the tree event stream representation S^{dynamic} following conditions hold:

1. all conditions from *simple tree event stream representation*
2. $e_i^{\text{start}}.node = e_j^{\text{end}}.node \Rightarrow i < j$
3. $e_i^{\text{end}}.node = e_j^{\text{end}}.node.parent \Rightarrow i < j$
4. $e_i^{\text{start}}.node = e_j^{\text{attrib}}.node \Rightarrow i < j$
5. $e_i^{\text{attrib}}.node = e_j^{\text{end}}.node \Rightarrow i < j$

As such, the dynamic tree event stream representation forms a superset of the simple tree event stream representation. In the following we assume each tree T to be available in one of the available types of tree event stream representations $S(T)$.

Identity Dimensions

The choice of a vertex' context determining its identity is limited in streaming environments. A vertex must be identifiable as it occurs in the stream, which is notably before any children are known. Furthermore, stream processing requires memory consumption to be limited. Thus only a limited history of the stream can be kept, preventing extensive backtracking.

Trivially considering a vertex $v \in \mathrm{V}(T)$ isolated from the original tree T discards all tree-relevant structural features. Hence, we choose the context of all vertices in a set of well-defined dimensions around each vertex. Ideally, dimensions represent orthogonal features of the tree. The identity of a vertex then only depends on vertices in these dimensions. The choice of the type and extent of these dimensions defines the precision of the identification.

The use of semi-structured data obviously marks the hierarchy as a notable feature. While the descendants of a vertex are not accessible in a stream, the ancestry is fully available. We, therefore, differentiate two independent dimensions that reflect the ancestry of a vertex:

1. The *parents* $\mathcal{P}(v)$ of a vertex v specify the vertex' global position in the hierarchy, and
2. the left *siblings* $\mathcal{Q}(v)$ of a vertex v address information about the local context.

Definition 4.12 (Order of parents)**.** Let T be a tree and a vertex $v \in \mathrm{V}(T)$. The *first order parent* $u \in \mathrm{V}(T)$ of v is the direct parent of v, that is $u = v.\,\mathrm{parent}$. Generalising this, the *n'th order parent* is therefore given by

$$\begin{aligned} p_1(v) &= \{u \mid u \in T, u = v.\,\mathrm{parent}\} \\ p_n(v) &= \{u \mid u \in T, u = p_{n-1}(v).\,\mathrm{parent}\} \end{aligned} \tag{4.1}$$

Definition 4.13 (Order of siblings)**.** Let T be a tree and a vertex $v \in \mathrm{V}(T)$. The *first order left sibling* u of v is the direct sibling to the left with $u.\,\mathrm{level} = v.\,\mathrm{level} - 1$. Generalising this, the *n'th order left sibling* is therefore given by

$$q_n(v) = \{u \mid u \in T(v.\,\mathrm{parent}) \wedge u.\,\mathrm{level} = v.\,\mathrm{level} - n\}, \tag{4.2}$$

where the same definition can also be used for right siblings by using $n < 0$.

Furthermore, the evolution over time given by the tree event stream representation as a whole defines another dimension of a vertex' ancestry. This adds a layer of concurrency, as it describes the change on multiple branches. We utilise a dimension to reflect the sequence of edit operations:

3. The *events* $\mathcal{S}(v)$ preceding a vertex v in the tree event stream reflect the concurrent context.

Definition 4.14 (Order of stream events)**.** Let $S(T) = (e_1, \ldots, e_k)$ be an event stream of a dynamic tree $\mathscr{T}$ at time k, that is $k = |\mathscr{T}|$. The *first order stream event* is the last known event e at time $k - 1$. Generalising this, the *n'th order stream event* is, therefore, given by

$$s_n = \{e \mid e \in S(T_k), e = e_{k-n}\}. \tag{4.3}$$

Notably, each of these dimensions is oriented towards the history of the tree event stream. It is thus safe to define the context of vertices as the extents $\mathcal{P}_i, \mathcal{Q}_j, \mathcal{S}_k$ in each dimension $\mathcal{P}, \mathcal{Q}, \mathcal{S}$. Based on this, we define the abstract identity of each single vertex $v \in T$ as $|\mathcal{P}(v), \mathcal{Q}(v), \mathcal{S}(v), \mathcal{V}(v)\rangle$.

4.4.4. Embedding Trees by Encoding Vertex Identities

There are three clearly distinguishable variants of the extent of each category $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{S}$: An extent of 0 excludes a dimension entirely. An upper bound of $n \in \mathbb{N}$ for the extent limits the dimension to a strictly confined neighbourhood. An unbounded extent includes the entire ancestry in a specific dimension.

Logically, the unbounded extent is an extreme limit of bounded extents. However, the two can be treated differently from an algorithmic point of view, and exhibit different fringe effects on identities.

Fixed-Length Encoding of Identities

For each of the given identity dimensions we can define an identity class with bounded length. We exemplary define an identity class on $\mathcal{P}$ dimension, that is

$$\begin{aligned} \mathrm{Id}_n^{\mathrm{p}}(v) &= |\mathcal{P}_p(v), \mathcal{Q}(v) = \varnothing, \mathcal{S}(v) = \varnothing, \mathcal{V}(v)\rangle \\ \mathcal{P}_p(v) &= |p_1(v), \ldots, p_p(v)\rangle, \end{aligned} \tag{4.4}$$

where p is the length of the path encoded in the identity.

A combined identity class uses several dimensions to build the identity of a vertex. For example, this allows us to directly express the pq-gram approach proposed by Augsten, Böhlen, and Gamper [21]. The authors employ a fixed-length encoding to define sub-tree patterns for tree decomposition: For each vertex, a fixed extent in $\mathcal{P}$ and the children of the vertex are considered. However, considering children as one dimension is not suitable for streaming environments, where children may occur at an arbitrary time after their parent. Thus, the approach of pq-grams is applicable neither to streaming environments nor dynamic trees. In the following, we demonstrate how the classical pq-grams can be adapted with the proposed formalisation to become suitable to streaming environments and dynamic trees.

Dynamic pq-grams The pq-gram approaches proposed in literature are not suitable to generate vertex identities on streams or dynamic trees. An inherent characteristic of the approach is the encoding of children for a given anchor vertex. However, in a streaming environment children for a given vertex are not known until each child vertex has occurred. Furthermore, a stream does not provide ahead of time the number of children to arrive, if any. Thus, the identity of a vertex is only well-defined after it has ended.

To enable pq-grams for dynamic trees, the context of the anchor vertex needs to be redefined. The original approach defines $p-1$ parents and q children as the context of the anchor vertex, as shown in Figure 4.7 on the following page. Our approach for dynamic pq-grams requires the anchor vertex to be the latest vertex of its context, which is defined by p parent vertices and $q-1$ left sibling vertices. Thus, the anchor vertex is shifted from vertex b to vertex e.

The classical pq-gram identity is subject to fringe effects. For example, on a specific layer, the leftmost vertex is part of only one pq-gram, whereas a center vertex is part of up to p pq-grams. To ensure the number of appearances for each vertex is not distorted by generated pq-grams, the authors introduce the pq-extended tree, which includes additional dummy vertices. This can be adapted to the representation of trees for dynamic pq-grams.

Definition 4.15 (Dynamic pq-extended tree [see 21, Definition 4.1 (pq-Extended Tree)])**.** Let T be a tree, and $p, q \in \mathbb{N} > 0$. The *dynamic* pq-*extended tree*, T^{pq}, is constructed from

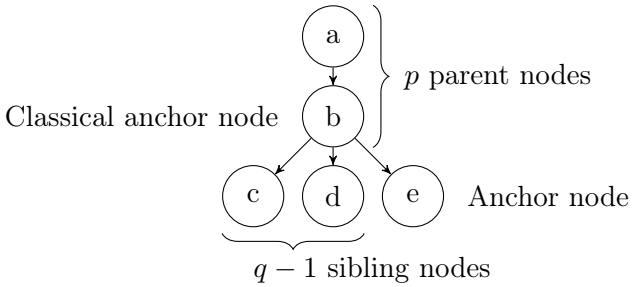


Figure 4.7.: Visualisation of *dynamic* pq-*grams* compared to original pq-grams. The original pq-gram approach uses the vertex denoted as *classical anchor vertex* to define pq-grams by accessing $p-1$ parent vertices and q child vertices. Dynamic pq-grams use the vertex denoted *anchor vertex* to specify the subtree structure by referencing p parent vertices and $q-1$ sibling nodes. Hence, dynamic pq-grams fulfil the condition to only access objects that are already known. That makes dynamic pq-grams feasible for streaming environments.

T by adding p ancestors to the root vertex, inserting $q-1$ vertices before any vertex that has no left siblings, inserting $q-1$ vertices after any vertex that has no right siblings, and adding q children to each leaf of T. All newly inserted vertices are dummy vertices that do not occur in T.

The choice of the anchor vertex ensures that dimensions are well defined as soon as an identity is needed. Adding virtual dummy vertices to the tree guarantees that each dimension extends sufficiently to match each real vertex equally often. In this context, we can define the dynamic pq-gram reliably.

Definition 4.16 (Dynamic pq-gram identity)**.** Let G be a dynamic pq-gram with anchor vertex $v \in \mathrm{V}(G)$. The *dynamic* pq-*gram identity is defined as*

$$\begin{aligned} \mathrm{Id}^{\mathrm{pq}}(v) :&= |\mathcal{P}_p(v), \mathcal{Q}_q(v), \mathcal{S}(v) = \varnothing, \mathcal{V}(v)\rangle \\ \mathcal{P}_p(v) &= |p_1(v), \ldots, p_p(v)\rangle \\ \mathcal{Q}_q(v) &= |q_1(v), \ldots, q_q(v)\rangle, \end{aligned} \tag{4.5}$$

where $p_p(v)$ denotes to the p'th order parent and $q_q(v)$ denotes the q'th order left sibling.

This makes the identity for dynamic pq-grams dependent only on a small context around each vertex. As such, it is suitable for stream processing, requiring only a well-defined neighbourhood around each vertex. However, it excludes any long and medium range effects. Every information outside its immediate neighbourhood is ignored for a vertex.

Definition 4.17 (Dynamic pq-gram identity profile [see 21, Definition 4.5 (pq-Gram Profile)])**.** Let T^{pq} be the dynamic pq-extended tree of a tree T and $S^{\mathrm{simple}}(T^{\mathrm{pq}})$ the respective tree event stream representation. The *dynamic* pq-*gram identity profile* $|T\rangle$ of T is defined as a multiset of all identities, that is

$$|T\rangle = \sum_{e \in S^{\mathrm{nested}}(T^{\mathrm{pq}})} \mathrm{Id}^{\mathrm{pq}}(e.node). \tag{4.6}$$

Following this definition, for each vertex that is appended to the tree, a new identity is generated. Each of those identities is appended to the current identity profile. As the identity profile is a collection of identities, the expansion of the identity profile can also be done in an incremental manner.

Complexity Given Property 4.4 on page 50 the upper bound of space complexity for finite-length encoding of identities is linear regarding the size of the tree event stream representation of a given tree.

Proof. Let m be the size of the tree event stream representation of a given tree T, with $m = |S(T)|$. For each event $e \in S(T)$ an identity Id is generated. Thus, in the worst case each identity Id is assigned to exactly one event e:

$$|S(T)| \geq \langle T|T \rangle.$$

Thus, the upper bound for space complexity for finite-length encoding strategies is $\mathcal{O}(m)$. □

However, the linear complexity is only worst case consideration for a fanout f with $f = 1$, which is not expected to be common in tree-structured data.

Theorem 4.18. Given a streaming environment with possibly infinite trees and a fanout $f > 1$, space complexity for fixed length encoding is constant regarding the given identity alphabet A, that is $\mathcal{O}(|A|)$.

To proof this statement, we exemplarily make the following assumptions:

Assumption 4.19. The identity class in use is Id^{p} thus $\mathcal{P}$ dimension is encoded at a fixed length p.

Assumption 4.20. The defining property of the identity is a vertex' label only.

Assumption 4.21. The labels of vertices are uniformly distributed.

Assumption 4.22. The tree event stream representation in use is simple tree event stream representation, thus for each vertex one event is processed.

Proof. Let f be the fanout of a tree T and A be the set of appropriate identities. The probability p to get a specific identity $a \in A$ from the tree event stream representation is given by $p = \frac{1}{|A|}$. Thus, the probability to not get this specific identity is $\frac{|A|-1}{|A|}$. Over the entire fanout f, the probability p_f to not include a specific identity is given by

$$p_f(a) = 1 - \left(\frac{|A| - 1}{|A|}\right)^f. \tag{4.7}$$

From this, the expectation value over all $a \in A$ can be derived as

$$E[F_i] = \sum_{a \in A} p_{f_i}(a). \tag{4.8}$$

The expectation value represents the expected number F_i of unique identities, when selecting f_i identities from A. This represents the number of vertices minus the number of collisions

within a given branch in the tree. Based on the number of unique identities, the compression c can be calculated as

$$c_i = \frac{F_i}{f_i}. \tag{4.9}$$

For each collision in a branch, the effective fanout for the descendants is increased due to merging of subtrees. For example, a collision of two identities results in an increased local fanout $f_i = 2f$. The local fanout f_i can be derived recursively as

$$\begin{aligned} f_0 &= f^0 \\ f_i &= \frac{f}{c_{i-1}} = \frac{f_{i-1} f}{F_{i-1}}. \end{aligned} \tag{4.10}$$

To derive the limits of the given behaviour, we evaluate border cases to show space complexity. First, we consider the expectation value for the number of unique identities:

$$E[F_i] = |A| \left(1 - \left(\frac{|A|-1}{|A|}\right)^{f_i}\right) \begin{cases} 1 & \text{if } f_i \to 1 \\ |A| & \text{if } f_i \to \infty \end{cases} \tag{4.11}$$

Notably, it follows that $E[F_i] \leq |A|$ and $E(F_i) \leq f_i$; the worst case of equality is given only for $|A| = 1$ or $f_i = 1$, respectively.

Finally, we consider the limits of the local fanout of the following branch, that is

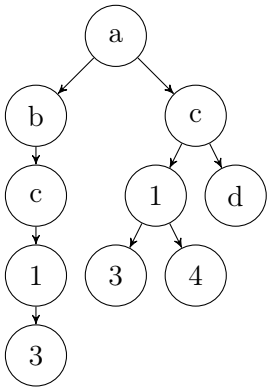
$$f_i = \frac{f_{i-1} f}{F_{i-1}} \begin{cases} 1 & \text{if } f \to 1 \\ \infty & \text{if } f \to \infty. \end{cases} \tag{4.12}$$

Since F_{i-1} is bounded, for a sufficiently deep tree we arrive at $f_i \propto f^i \propto |T|$. In general, by the pigeonhole principle [107] the probability to get a collision reaches 1 whenever $f_i > |A|$. In practice, this means that as the depth of the tree increases, the chance of picking an unused identity from A approaches 0. Consequently, we have a constant complexity for space, that is $\mathcal{O}(|A|)$. □

The space complexity of identities of fixed-length encoding strategy is constant regarding the size of the alphabet of identities. This is due to the alphabet of identities forming an upper bound on the number of distinct vertices. However, this means that the information available from an identity profile of identities is bounded as well. In effect, this means the benefit of complexity comes at the cost of sensitivity.

Diamonds in Tree Embeddings The classical pq-gram approach measures the similarity of two trees by determining the number of overlapping pq-grams. The discriminating feature is, therefore, the identity of each single sub-tree independent from its position in the tree. This lack of position is a severe contrast to our assumption that the hierarchy of vertices is a defining feature (see Section 3.2.1 on page 29). Thus, we have studied the pq-grams with respect to the implications of their limited hierarchical information for our specific use case.

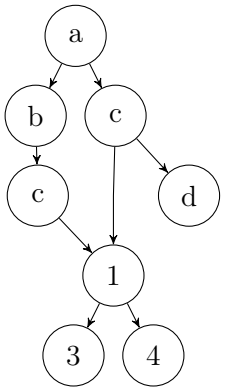
The choice of parameters is integral to balance the expressiveness of the local vertex context against the cost of buffering a large neighbourhood. Identifying vertices with only a small context carries the risk of mapping distinct positions across the tree to the same identity. While this can be advantageous to find small similarities in distinct hierarchies,



(a) Example tree

Tree Layer	Dynamic pq-grams
1	(∗, ∗, a)
2	(a, ∗, b), (a, b, c)
3	(b, ∗, c), (c, ∗, 1), (c, 1, d)
4	(c, ∗, 1), (1, ∗, 3), (1, 3, 4)
5	(1, ∗, 3)

(b) Generated dynamic pq-grams per tree layer for $p = 1$ and $q = 2$



(c) Projection based on generated dynamic pq-grams

Figure 4.8.: Visualisation of the diamond-building process by applying dynamic pq-grams. The example is based on an example tree given in Figure 4.8a. Figure 4.8b lists relevant dynamic pq-grams that are created with configuration $p = 1$ and $q = 2$. Calculated dynamic pq-grams that contain dummy vertices are excluded from the visualisation for simplicity. Based on those dynamic pq-grams, Figure 4.8c visualises the appropriate projection. The vertex labelled with 1 constitutes a *diamond*. The visualisation of this projection reveals, that the vertices labelled 2, 3 and 4 cannot be associated unambiguously to their affiliated parent branch. Thus, distance functions based on this embedding calculate distorted results.

it sacrifices information required to compare extended hierarchies. Improperly mapped identities can create tree embeddings that distort tree distances. This distortion can either result in trees that are considered more similar than expected or more distant than expected.

The increased similarity of tree embeddings for distinct trees is a result of the tree decomposition approach itself. If the decomposition of a tree is improperly chosen, distinct parts of the tree result in an identical representation even though representing distinct features of the tree. This effect hereinafter called the diamond-building characteristic, occurs for several decomposition methods; since our use case relies on hierarchical data, this effect cannot be neglected. In the following, we analyse this diamond-building characteristic and deduce embedding approaches for static and dynamic trees that minimise this negative impact.

Decomposition methods which consider a specific number of vertices, for example, a fixed number of parent or child vertices, to describe the local context of a vertex are vulnerable to distortions. Let us consider the dynamic pq-grams for example. Its parameters p and q define a context around each vertex, which can be interpreted as a shape of subtrees the tree is decomposed to. Each vertex $v \in \mathrm{V}(T)$ is identified by a collection of such subtrees, which contain p parent vertices as well as q children vertices around v. Depending on the parameters of the subtree shape, the logical representation of several subtrees may form *diamonds*. A diamond constitutes a vertex that has more than one parents, that is an undirected cycle. This representation, therefore, translates back to a Directed Acyclic Graph (DAG) instead of a tree.

Figure 4.8 visualises this diamond-building effect. The example considers dynamic pq-grams for tree decomposition with configured parameters $p = 1$ and $q = 1$. Figure 4.8b on

the preceding page shows the corresponding snippets in breadth-first order. Both vertices labelled with 3 encode the same parent vertex. When reconstructing the graph from its logical representation, the original parent cannot be unambiguously determined. Therefore, only a diamond structure resolves the information contained in the decomposition (compare Figure 4.8c on the previous page).

Example 4.23. Consider the vertex labelled 4 from Figure 4.8c on the preceding page. The vertex is a descendant of a *diamond* that is located in the vertex labelled 1. There are two choices to which parent branch it might belong to; either to the branch identified by the paths $[a, b, c, 1]$ or $[a, c, 1]$. Thus, there can be different trees resulting in the same logical representation for their given tree embeddings. This, in turn, results in a higher similarity for trees featuring either possibility.

Definition 4.24 (Diamond)**.** A *diamond* at a given path length n is defined by two vertices $u, v \in \mathrm{V}(T)$ that share the same identities in $|\mathcal{P}_n\rangle$ but differ in $|\mathcal{P}_{n+1}\rangle$, that is

$$\mathrm{dia}(v, T) = \{u \,|\, \mathcal{P}_n(v) = \mathcal{P}_n(u) \wedge \mathcal{P}_{n+1}(v) \neq \mathcal{P}_{n+1}(u)\}, \forall u \in \mathrm{V}(T), \tag{4.13}$$

where $\mathcal{P}_n(v)$ is the sequence of n parent vertices of vertex v.

This definition holds true for all identities that are generated using tree decomposition methods. The above definition specifically focuses on the parent-child relationship $|\mathcal{P}_n\rangle$ within trees. This is a special case for our current considerations regarding dynamic pq-grams. In general, diamonds can have an effect on any identity using bounded extents of dimensions. Without loss of generality, we concentrate on the more specific Definition 4.24 in $|\mathcal{P}\rangle$ and showcase its consequences regarding dynamic pq-grams.

Example 4.25. Based on Figure 4.8a on the preceding page, we show the application of dynamic pq-grams. For simplicity, the value of q is fixed to 1, excluding any sibling vertices. Given this, the following identities are generated for the vertices $u, v \in \mathrm{V}(T)$ for $p = 1, 2, 3$:

$$\begin{aligned}
&\mathcal{P}_1(v) = |\text{"1"}\rangle, && \mathcal{P}_1(u) = |\text{"1"}\rangle && \Leftrightarrow \mathcal{P}_1(v) = \mathcal{P}_1(u)\\
&\mathcal{P}_2(v) = |\text{"1", "c"}\rangle, && \mathcal{P}_2(u) = |\text{"1", "c"}\rangle && \Leftrightarrow \mathcal{P}_2(v) = \mathcal{P}_2(u)\\
&\mathcal{P}_3(v) = |\text{"1", "c", "b"}\rangle, && \mathcal{P}_3(u) = |\text{"1", "c", "a"}\rangle && \Leftrightarrow \mathcal{P}_3(v) \neq \mathcal{P}_3(u)
\end{aligned}$$

For the given tree the application of dynamic pq-grams with $p = 0$ or $p = 1$ will result in identities creating a diamond, $\mathrm{dia}(u, T) = \mathrm{dia}(v, T)$. A choice of $p \geq 3$ is required to avoid diamonds.

Overlapping Diamonds A diamond is the consequence of two distinct vertices sharing the same identity. Extending on this, it is possible for several diamonds to overlap if multiple distinct vertices share the same identity. Overlapping diamonds, in turn, amplify the bias of the resulting distance approximation.

Therefore, to determine the bias that a diamond may induce for the calculated distance value, the level of a diamond needs to be considered. We define the level of a diamond as the number of initially distinct vertices that are merged into the diamond.

Definition 4.26 (Level of a Diamond)**.** The *level of a diamond* $\mathrm{L}^{\mathrm{diamond}}(v, T)$ is defined by the number of vertices $|u| \in \mathrm{dia}(v, T) \setminus \{v\}$ that share the identity of v in $|\mathcal{P}_n\rangle$, that is

$$\mathrm{L}^{\mathrm{diamond}}(v, T) = \max(0, |\,\mathrm{dia}(v, T)| - 1), \tag{4.14}$$

where max is a function that returns the maximum of the given values.

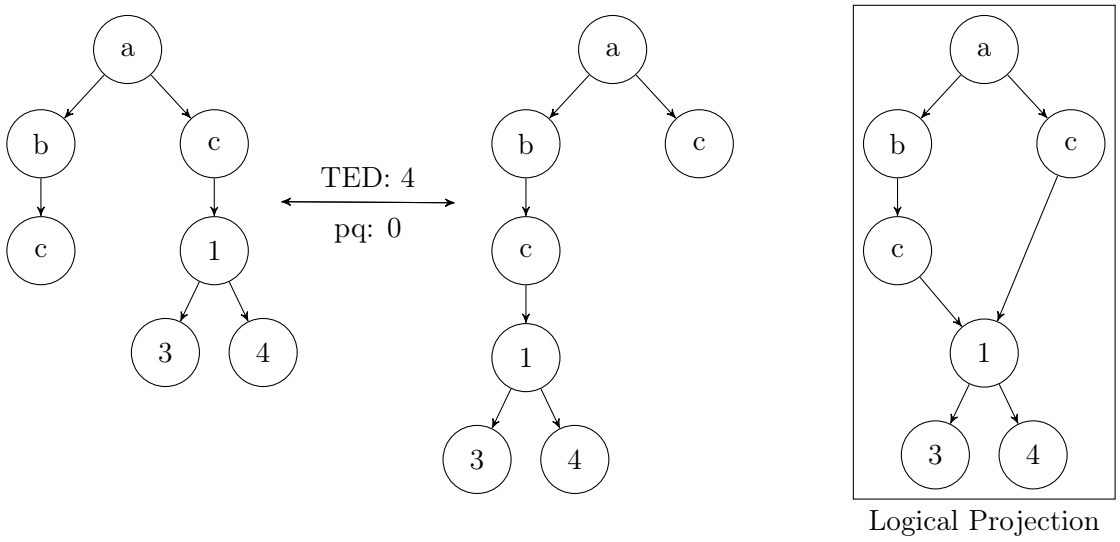


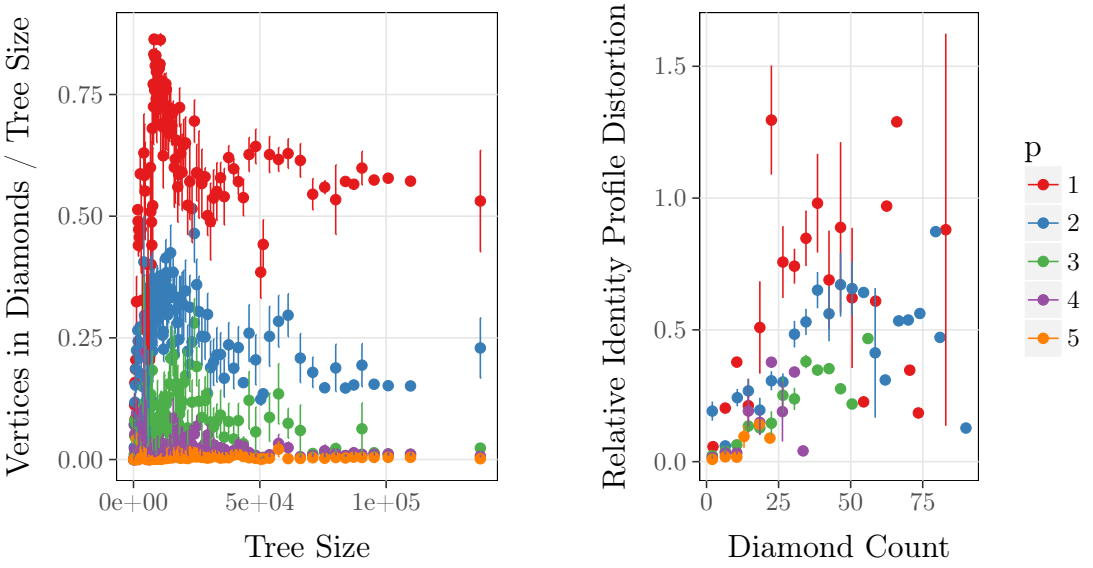
Figure 4.9.: Comparison of Tree Edit Distance and pq-gram distance. The pq-gram distance recognises both trees as equivalent. The reason for this is visualised in the logical projection. It can be seen that a diamond is created in the vertex labelled 1. However, minimum number of edit operations regarding Tree Edit Distance is 4. Consequently, diamonds cause trees to be recognised as more similar than expected.

Following this definition, a diamond with level 1 is built from two distinct vertices that share the same identity. Each further vertex that maps to the same identity that caused the diamond-building process increases the diamond level by one.

Impact of Diamonds The occurrence of diamonds has several implications on the accuracy of tree representations: First, the encoding of tree sub-structures from different positions within the tree is not distinct. This results in higher similarity for trees with differing hierarchical features but similar tree sub-structures. Figure 4.9 visualises this feature. Although the TED suggests a distance of 4, the resulting distance based on given dynamic pq-gram is 0. Second, descending vertices of a diamond might unintentionally be associated to the wrong branch. This allows for sub-structures to be mixed across branches. This is visualised in Figure 4.8 on page 59. The vertex labelled 4 in Figure 4.8a is unambiguously related to the path $[a, c, 1]$. After a projection utilising dynamic pq-grams (see Figure 4.8c), the vertex is a descendant of the diamond in vertex labelled 1. Therefore, the relation to its initial branch is no longer possible. Instead, the vertex is either related to the path $[a, b, c, 1]$ or $[a, c, 1]$.

A diamond is the result of a collision of identities from distinct branches. The child vertices of the diamond are the child vertices of each distinct branch. This means that each child of a diamond can be considered a second-degree child of *every* parent branch. Following this, the distortion grows linearly with both the number of vertices within subtrees of a diamond and the number of parent branches.

For each diamond $\{v \mid L^{\text{diamond}}(v, T) > 0\}, \forall v \in \mathrm{V}(T)$, the vertices of its subtree $\mathrm{V}(\mathrm{T}(v))$ cannot unambiguously be attributed to the correct parent branch. As a worst case estimate, we can consider all vertices of the subtree, that is $|\mathrm{V}(\mathrm{T}(v))|$, as wrongly assigned to a



(a) Proportion of vertices that create diamonds

(b) Relative distortion for varying number of diamonds

Figure 4.10.: The Figures 4.10a to 4.10b on this page show the influence of diamonds for dynamic pq-grams for different values of p, $p = 1, \ldots, 5$. The value of q is fixed to 1 to specifically analyse the influence of p on the diamond creation process. The analysis is applied to batch system jobs from GridKa. For varying sizes of batch system jobs ten samples are randomly selected. Figure 4.10a on the left visualises the percentage of vertices of the given batch jobs that create diamonds for calculated tree projection. Figure 4.10b visualises the relative distortion caused by the induced diamonds compared to the number of vertices. The analysis shows, that the influence of diamonds decreases by increasing the value of p.

parent branch. The number of parent branches corresponds to the number of vertices forming a diamond, that is $L^{\text{diamond}}(v, T) + 1$.

Therefore, the distortion of a tree embedding for a tree T caused by diamonds can be estimated as

$$\epsilon^{\text{diamond}}(T) = \sum_{v \in \mathrm{V}(T)} |\{w \,|\, w \in \mathrm{V}(\mathrm{T}(u)), \forall u \in \mathrm{dia}(v, T) \setminus \{v\}\}|. \tag{4.15}$$

This estimation provides an upper bound of the induced distortion to the tree embedding caused by diamonds. It also reflects the distortion caused by *nested diamonds*. A nested diamond is a diamond whose vertices $\mathrm{V}(\mathrm{T}(u)), L^{\text{diamond}}(u, T) > 0$ are a subset of vertices of another diamond $\mathrm{V}(\mathrm{T}(v)), L^{\text{diamond}}(v, T)$ of a given tree T, that is $\mathrm{V}(\mathrm{T}(u)) \subset \mathrm{V}(\mathrm{T}(v)), \mathrm{dia}(u, T) \cap \mathrm{dia}(v, T) = \varnothing$.

Figure 4.10 visualises the effects of diamonds in the recorded batch job data. This analysis shows two distinct effects of diamonds concerning our data: First, given sufficiently large trees, the number of diamonds depends only on p. A significant value of p, compared to the height of the given tree, allows suppressing the occurrence of diamonds. Second, the distortion caused by diamonds mostly depends on the number of diamonds. Reducing

the distortion requires the suppression of diamonds. Also, large values of p also impose an upper bound on the distortion.

As shown above, diamonds induce distortion in the tree embeddings and thus also on the distance calculation. This distortion is linearly dependent on the actual level of the diamond and promotes similarity between different hierarchies. Therefore, the existence of diamonds implies an unwanted bias for distance measurement on a large scale but enables the matching of substructures.

When considering dynamic pq-grams to measure distances between trees, diamonds can hide veritable differences. The lack of a substructure at one position can be offset by a similar structure at another position. Excluding this in a non-trivial way without prior knowledge about the underlying data is impossible. Safe values for p and q can be derived from a dedicated analysis, but this does not guarantee the absence of diamonds beyond the analysed dataset. An analysed sample may fail to represent all relevant cases, and data might evolve over time, especially in streaming environment. Therefore, a repeated analysis of data is required to limit the existence of diamonds.

Still, the presence of diamonds may provide desirable advantages. Collisions due to diamonds reduce the number of unique identities. This, in turn, reduces memory utilisation as only few identities are stored, and a higher multiplicity does not require additional space. For $k = |\{\mathrm{dia}(v,T) \,|\, v \in T \cap L^{\mathrm{diamond}}(v) > 0\}|$ diamonds memory requirement is at $\mathcal{O}(k)$ whereas a method that prevents diamonds requires memory of $\mathcal{O}(kl)$, with $l = max(L^{\mathrm{diamond}}(v), \forall v \in \mathrm{V}(T))$. Thus, the memory requirement is inversely proportional to the average level of diamonds.

While our use case specifically relies on the entire hierarchy of data, diamonds may be desirable for partial matching. While collisions inside a tree reduce information, collisions across two trees reveal matching substructures. This can be desirable if the relevant content is enclosed in an irrelevant hierarchy [23, 25].

Handling To identify similarities inside trees, an estimate for the scale of similarities is required. While this approach is limited in sensitivity, its complexity is guaranteed to be bound. In contrast, to accommodate use cases that rely on the global topology of a tree, it is necessary to eliminate diamonds entirely by encoding all parents of a vertex. Thus, we need to derive an identity class that avoids diamond-building effects efficiently.

Infinite-Length Encoding of Identities

Using finite-length encoding of identities, extents of dimensions must be tuned for every use case. Otherwise, tree decompositions can distort tree structure information, biasing distances to be more similar. While large extents reduce this effect, they require a large neighbourhood to be tracked for each vertex.

To remedy this, we propose a recursive encoding strategy that builds the identity of each vertex from all of its parent vertices. Thus, an identity includes all vertices up to the root vertex. We call this an infinite-length encoding of a specific identity dimension since it implies an infinite extent of the corresponding dimension. However, this limit allows for computational and algorithmical optimisation.

Root Path Encoding In analogy to fixed-length identity classes (see Section 4.4.4 on page 55), we also propose an infinite-length encoding for any given identity dimensions.

Algorithm 1 Recursive infinite-length identity encoding

Precondition:
`v` is a vertex which *identity* is determined, $v \in \mathrm{V}(T)$
$\mathtt{i}_u$ is the *identity* of $u \in \mathrm{V}(T)$, where $u = v.parent$
Postcondition: `identity` of vertex v

```
1: function IDENTITY(v, i_u)
2:     if i_u ≠ ∅ then
3:         identity ← new identity(i_u, v.lbl)
4:     else
5:         identity ← new identity(∅, v.lbl)
6:     return identity
```

Again, we exemplary define an identity class on $\mathcal{P}$ dimension, which is a special case of Equation (4.4) as

$$\begin{aligned} \mathrm{Id}^{\mathrm{P}}(v) &= |\mathcal{P}_\infty(v), \mathcal{Q}(v) = \varnothing, \mathcal{S}(v) = \varnothing, \mathcal{V}(v)\rangle \\ \mathcal{P}_\infty(v) &= |p_1(v), \ldots, p_k(v)\rangle, k = v.depth. \end{aligned} \tag{4.16}$$

While logically equivalent to a finite-length encoding of sufficient depth, the infinite-length encoding strategy is advantageous for calculating identities. Algorithm 1 summarises the proceeding. To calculate the actual identity of a vertex $v \in \mathrm{V}(T)$, not all vertices on its path to the root vertex have to be considered. By recursively encoding the identity of the parent vertex, considering only the direct parent $v.parent$ of each vertex v is sufficient. Thus, the identity calculation complexity is reduced to $\mathcal{O}(1)$.

In $\mathcal{P}$ dimension two vertices $v, w \in \mathrm{V}(T)$ share the same identity whenever $v.\,\mathrm{lbl} = w.\,\mathrm{lbl}$ and $v.\,\mathrm{parent} = w.\,\mathrm{parent}$. This implies lossy compression, but with different features compared to finite-length identity classes. Most importantly, no diamonds can occur in an infinite-length dimension (see Section 4.4.4 on page 58). Thus, finite-length encoding compresses trees with repetitive occurrences of vertices in the same layer of the hierarchy. As this identity class only considers one dimension, it is insensitive for the order of vertices. However, applying infinite-length encoding in several dimensions, especially $\mathcal{S}$ dimension, allows for an unbounded number of identities, even if the underlying alphabet of labels is limited.

Sibling Root Path Encoding In analogy to the dynamic pq-grams approach, we combine infinite-length encoding in $\mathcal{P}$ with finite-length $\mathcal{Q}$:

$$\begin{aligned} \mathrm{Id}_q^{\mathrm{Pq}}(v) &= |\mathcal{P}_\infty(v), \mathcal{Q}_q(v), \mathcal{S}(v) = \varnothing, \mathcal{V}(v)\rangle \\ \mathcal{P}_\infty(v) &= |p_1(v), \ldots, p_k(v)\rangle, k = v.depth \\ \mathcal{Q}_q(v) &= |q_1(v), \ldots, q_q(v)\rangle, \end{aligned} \tag{4.17}$$

where q is the number of left siblings of $v \in \mathrm{V}(T)$ to retain for the specified identity class.

This identity class implements a mixture of different characteristics. It imposes a finite-length ordering of siblings; this can be seen as a well-defined context of short range as given by q. This emphasises local information, independent from the hierarchy. In contrast, the infinite-length ordering of parents ensures an absence of diamonds. This minimises distortion of the hierarchy.

Complexity Based on the considerations for constant space complexity for fixed-length encoding we derive similar space complexities for infinite-length encoding. In general, the infinite-length encoding also exhibits constant behaviour regarding the alphabet of identities (compare Section 4.4.4 on page 57). The difference to consider for complexity is the identity class itself, as it dictates the size of the alphabet of identities.

Given an alphabet of labels Σ_λ, the maximum alphabet of identities is limited to $|A| \leq |\Sigma_\lambda|^{p+q+s}$, where p, q, and s are the considered lengths of each dimension $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{S}$ respectively. However, with infinite-length encoding, this limit depends on the structure of the tree imposing *effective* limits on each dimension.

Theorem 4.27. Infinite-length encoding schemes in $\mathcal{P}$ dimension have space complexity of $\mathcal{O}(n^{\frac{\log|\Sigma_\lambda|}{\log f}})$, meaning that it is sublinear or even constant depending on the fanout f.

Let us consider the following assumptions:

Assumption 4.28. The identity class in use is Id^{P}, with the $\mathcal{P}$ dimension encoded up to the root level.

Assumption 4.29. The tree is a complete tree with a fanout of f.

Proof. Based on the given assumptions, the root-level encoding is bounded by the depth of the tree. For a complete balanced tree, the depth can be derived as $d_{\max} = \log_f(n+1)$ for a given number of vertices $n = |T|$.

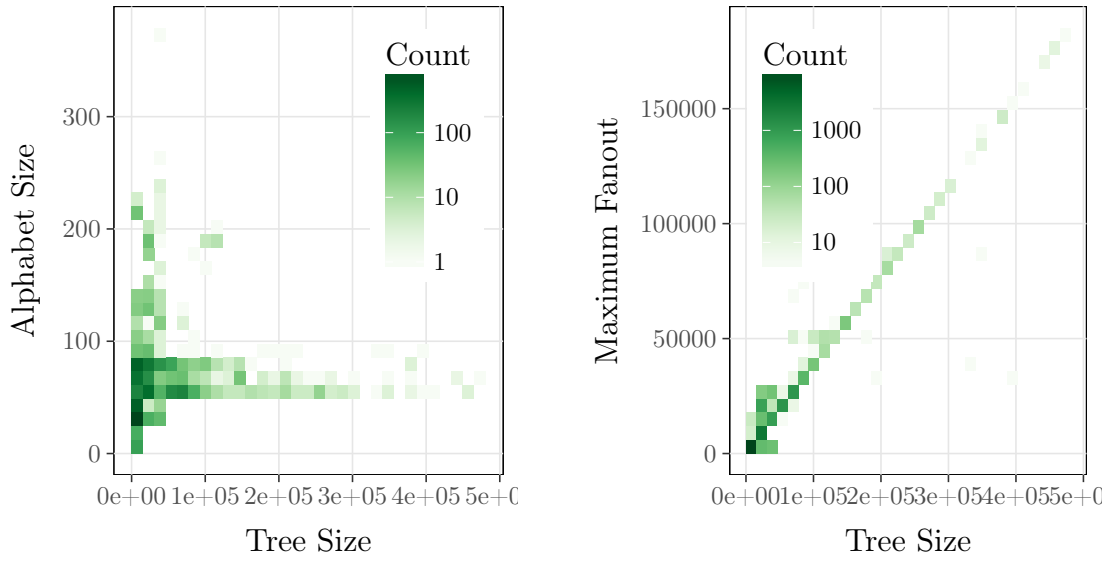
$$\begin{aligned} |\Sigma_\lambda|^{d_{\max}} &= |\Sigma_\lambda|^{\log_f(n+1)} \\ &= (n+1)^{\log_f|\Sigma_\lambda|} \\ &= (n+1)^{\frac{\log|\Sigma_\lambda|}{\log f}} \end{aligned} \tag{4.18}$$

By definition, the number of unique identities can never exceed the number of vertices. Thus, we can derive the space complexity of

$$\mathcal{O}(n^{\bar{f}}) \qquad \text{with } \bar{f} = \min\left(1, \frac{\log|\Sigma_\lambda|}{\log f}\right).$$

For all trees with $f > |\Sigma_\lambda|$ space complexity is guaranteed to be sublinear, even if the depth of the tree is unbounded. As shown in Equation (4.11), even for $f < |\Sigma_\lambda|$ we can expect an effective label alphabet $E[F_i]$ smaller than f and thus sublinear behaviour. □

In reality the frequencies of vertex labels are not uniformly distributed. For our given use case, where labels are the names of executed processes, few selected labels appear more frequently than others. Empirical analysis, see Figure 4.11 on the following page shows that this further improves the expected storage requirements. While for many trees the alphabet size is constant with an increasing number of vertices of the given trees (compare Figure 4.11a on the next page) the maximum fanout also grows linearly with the number of vertices in the tree (compare Figure 4.11b on the following page). Thus, we can expect even better compression for growing trees.



(a) Alphabet size for different tree sizes.

(b) Maximum fanout for different tree sizes.

Figure 4.11.: Count on number of distinct process names and maximum fanout in a real HEP use case for different tree sizes. The analysis shows, that the underlying alphabet is small compared to the number of vertices within a given tree. Figure 4.11a also shows, that we deal with at least two different classes of workflows. One class even has a constant alphabet size even for increasing vertex counts. Analysis of the maximum fanout shows that it is proportional to the tree size. Thus, good compression ratios can be expected for our given use case.

4.4.5. Summary

To process trees in a streaming environment, we employ a decomposition-based approach: Each vertex and its neighbourhood are compressed to an identity, independent of other vertices. This approach is applicable both to static and dynamic trees.

Instead of choosing an arbitrary neighbourhood, we use well-defined dimensions centred on each vertex. Thus, all identity classes exhibit similar features, while the extent of the neighbourhood in each dimension can be adjusted as required. Most importantly, the dimensions we have defined make identity classes based on them suitable for stream-based analyses.

Using a fixed extent of dimensions incurs a tradeoff between preserving hierarchical information and reducing algorithmic and space complexity. There is a direct correspondence between the dimensional extent of identities and their space complexity. The more dimensions and the higher the extents used to generate an identity, the higher is the worst case space requirement, as the number of combinations of primitive labels increases exponentially.

Thus, we allow for all dimensions to be considered infinite-length extents. This is algorithmically advantageous, being comparable to the best case of finite-length. While space complexity is worse, we can guarantee a worst case of $\mathcal{O}(n)$ even for degenerated trees. As shown, even for large label alphabets, the average complexity is sublinear, and constant complexity is possible even for infinite trees.

4.5. Tree Distances

Tree distances derive the difference between two trees, creating a scale to compare different trees with each other. As trees are complex data structures, differences can appear on multiple levels – be it the tree hierarchy, vertex features or even abstract properties such as symmetry. Since each of these can be expressed and weighted differently, there is no single *true* tree distance. Our distance approach is based on leveraging vertex features and the structural information encoded by identities to incrementally compare regions of trees.

To make a tree distance useful in the domain of dynamic streaming trees, it needs to satisfy the following requirements:

- *Scalability*: Algorithms to measure tree distances must produce results in linear time with small constant factors or even sublinear time. Furthermore, memory utilisation must also take this requirement into account. These limitations on computation and memory complexity are needed due to the unconstrained size of dynamic trees in streaming environments. This is especially true in the context of HEP batch systems (see Section 2.3 on page 20).

- *Sensitivity*: Tree distance algorithms need to be more sensitive to changes in high-quality vertices and their properties. Changes in low-quality vertices can be considered less important and should, therefore, have less impact on distance results. In general, changes that imply an impact on a bigger proportion of vertices within a tree should also have a bigger impact on results.

- *Coverage*: Tree distance algorithms need to support both the topology of a tree as well as the different properties of vertices. Topological changes include introduction of new vertices, removal of existing vertices, and changes in vertices, or their ordering. Properties of a vertex can refer to the vertex itself, such as its lifetime, or properties associated with the vertex.

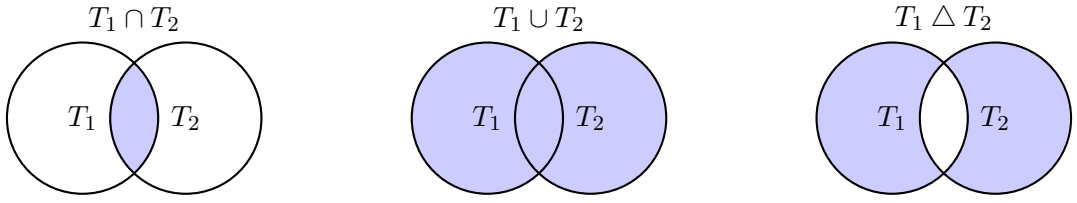
Logically, our approach is based on a projection of one tree identity profile onto another. The overlap of both serves as a measure of similarity, whereas the differences provide the distance. While this guarantees only a pseudo-metric for trees, it is a metric on the identity profiles. In fact, if identity profiles encode all relevant tree features, our approach is a metric for trees as well.

For the purpose of comprehensiveness and completeness of the given formalisation, we consider insertion-only trees first. At this level, our approach is akin to common distance measures for static trees, such as TED. However, this is only for simplicity, as each vertex is considered only once in this setup. The general approach is later applied to dynamic trees as well.

4.5.1. Identity Profile Projection

While our approach is applicable for various types and formats of trees, introducing it at a restricted level allows drawing parallels to established distance measures. At the scope of static trees without attributes, our approach can be outlined as a vertex-wise matching of identities. This is analogous for example, to the vertex-wise cost model of TED, and the counting of individual pq-grams.

We express static trees as a stream of insertion-only events (see Section 4.4.3 on page 52). Each event thus corresponds to exactly one vertex of the tree, producing one identity per



(a) Intersection of two trees (b) Union of two trees (c) Symmetric difference of two trees

Figure 4.12.: Visualisation of the operations intersection, union, and symmetric difference regarding collections of identities for vertices in two trees T_1 and T_2. The intersection of two trees visualised in Figure 4.12a includes all identities that both trees T_1 and T_2 have in common. The union of T_1 and T_2 depicted in Figure 4.12b comprises the sum of all identities. Finally, the symmetric difference is the disjunctive union of the trees, see Figure 4.12c. That is all identities that are in either of the collections and not in the intersection of T_1 and T_2.

vertex. Without loss of generality, one can assume that the identity profile of the tree could be stored statically, such as a set, multiset, or list.

The distance of two static trees $\mathscr{T}_1$ and $\mathscr{T}_2$ is the number of vertices belonging to only one of either trees. We can express this distance by the *symmetric difference* of the identity profiles of the two trees. This is the number of identities that differ between the respective identity profiles. Related to this, we can express the similarity between both trees as the *union* of the identity profiles. This concept of symmetric difference and union is visualised in Figure 4.12.

Recall that we have already defined the similarity of two vertices v, w based on their identity as their *projection* $\langle v|w\rangle$. Similarly, the intersection of identity profiles can be expressed as a projection of one identity profile onto the other. Thus, we extend our previously introduced framework supporting the identity profile projection of identities by the identity profile projection of identity profiles.

For the projection of two identities, we distinguish between a baseline identity, denoted as *recorded* identity, and an observed identity. This is reflected by our notation: the bra $\langle ...|$ holds the recorded identity, the ket $|...\rangle$ holds the observed identity. Thus, a recorded identity of $\mathscr{T}_1$ is given by $\langle \mathcal{P}, \mathcal{Q}, \mathcal{S}, \mathcal{V}|$, and each observed identity of $\mathscr{T}_2$ is given by $|\mathcal{P}_2, \mathcal{Q}_2, \mathcal{S}_2, \mathcal{V}_2\rangle$.

Throughout this work, we consider the identity of a vertex to be an unambiguous identifier. That means two identities either match exactly, or not at all[2]. This feature of an identity projection is defined as:

$$\begin{aligned}\langle \mathcal{P}, \mathcal{Q}, \mathcal{S}, \mathcal{V} \,|\, \mathcal{P}_2, \mathcal{Q}_2, \mathcal{S}_2, \mathcal{V}_2\rangle &= \langle \mathcal{P} \,|\, \mathcal{P}_2\rangle\langle \mathcal{Q} \,|\, \mathcal{Q}_2\rangle\langle \mathcal{S} \,|\, \mathcal{S}_2\rangle\langle \mathcal{V} \,|\, \mathcal{V}_2\rangle \\ &= \delta_{\mathcal{P}\,\mathcal{P}_2}\delta_{\mathcal{Q}\,\mathcal{Q}_2}\delta_{\mathcal{S}\,\mathcal{S}_2}\delta_{\mathcal{V}\,\mathcal{V}_2}, \end{aligned} \tag{4.19}$$

[2]This is largely motivated by efficiency of implementation. An exact match can be looked up and calculated at $\mathcal{O}(1)$ across multiple trees.

where δ_{ij} is the Kronecker delta that is defined as

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases} \tag{4.20}$$

Recall that the identity profile of T_1 may have arbitrary order for contained identities (see Section 4.4.2 on page 50). Instead, order can be encoded into the identities to express parent, sibling or stream order (see Section 4.4.3 on page 54). We extend the identity projection to a identity profile of a tree T_1 as the recorded part and an identity of a vertices of a tree T_2 as the observed part:

$$\langle T_1 |\, \mathcal{P}_2(v), \mathcal{Q}_2(v), \mathcal{S}_2(v), \mathcal{V}_2(v) \rangle, \forall v \in \mathrm{V}(T_2) \tag{4.21}$$

Each identity in the identity profile of T_2 is individually compared for similarity with the entire identity profile of T_1. This piecewise comparison of identities is compatible with T_2 being a dynamic tree in a streaming environment.

Example 4.30. The comparison of identities can be implemented with a hash function, with each recorded identity stored in a hash table. Thus, when receiving an identity from an observed tree, the hash value of this identity can be checked for collision in the hash table of the identity profile of T_1. A collision indicates that the identity is not part of the recorded tree, and the two trees differ.

In the context of trees, there is no definitely correct aggregation of multiple, individual identities. Tree distance approaches in literature correspond to a range of possible aggregation semantics.

Example 4.31. Consider a set to store identities of identity profiles. Each identity occurs only once in the set. Therefore, the cardinality of identities cannot be considered in the tree distance. In contrast, using a multiset to store identities preserves cardinality.

At this point, the comparison to common, existing implementations suggests a simple counting of identities [23, 25, 49, 151, 233]. However, we will later on use less naive semantics, taking into account features of the corresponding vertices as well. To reflect this, we introduce a formal identity profile projection operator θ that expresses the semantics between vertices with the same identity.

Definition 4.32 (Identity Profile Projection)**.** Let $|\mathscr{T}_1\rangle$ be an identity profile of a tree $\mathscr{T}_1$ and $|\mathscr{T}_2\rangle$ an identity profile of a tree $\mathscr{T}_2$. The *identity profile projection* of the two given identity profiles is defined by

$$\langle \mathscr{T}_1 | \theta | \mathscr{T}_2 \rangle,$$

where θ is an identity profile projection operator that is applied as a statistic to the underlying collection of identities.

Profile Projection Operator

The use of a identity profile projection operator introduces the capability to describe relations between individual vertices, which are themselves described via identities. This is analogous for example, to the global optimisations of individual operations required by

Tree Edit Distances [41]. However, the applicability as stream-based distance measurement puts constraints on capabilities.

Most importantly, an identity profile projection operator is limited to the scope of a single vertex or few vertices at once. This rules out global optimisations, or the invalidation of earlier decisions. However, information of the identity profile can be used even in streaming environments. As shown in Section 4.4 on page 49, the space complexity of identities converges to a constant factor for an adequate choice of identity class. This makes it possible to preserve information at the granularity of identities. This is analogous to sketching, which can be used complementary.

The most basic identity profile projection operator we consider is based on set theory. This identity profile projection operator determines the overlap of two identity profile projections as the overlap of their sets of identities. We introduce the identity profile projection operator 1 based on the definition of the indicator function [66]. Thus the identity profile projection operator $1 : |v \in \mathrm{V}(\mathscr{T}')\rangle \to \{0, 1\}$ is defined as

$$\langle \mathscr{T}_1|1|v\rangle = \begin{cases} 1 & \text{if } \langle \mathscr{T}_1|v\rangle = 1 \wedge \langle \mathscr{T}_2|v\rangle = 0 \\ 0 & \text{if otherwise.} \end{cases}$$

This identity profile projection operator indicates unique membership of an identity in a given identity profile. Consequently, $\langle \mathscr{T}_1|1|\mathscr{T}_2\rangle$ determines the number of unique identities shared between the two given trees $\mathscr{T}_1$ and $\mathscr{T}_2$.

As identities compress similar vertices, it is likely and often desirable to have multiple vertices with the same identity. Yet, testing the presence of identities with the identity profile projection operator 1 ignores differences in the number of similar vertices due to compression. Thus, the identity profile projection operator may assume two trees to be equal although their cardinality of vertices might differ. For a correspondence of tree distance with the tree size, one needs to consider the multiplicity of identities.

Therefore, we consider the multiplicity identity profile projection operator $M : U \to \mathbb{N}$, that is $\langle \mathscr{T}_1|M|\mathscr{T}_2\rangle$. This identity profile projection operator M ensures the overlap of two given identity profiles based on their multiplicity of distinct identities. This ensures to know about the set of vertices $u' \in \mathrm{V}(\mathscr{T}_2)$ that are related to the same identity to determine its multiplicity.

4.5.2. Static Projection Distance

The identity profile projection of identity profiles provides the absolute similarity of trees. When analysing multiple trees of different sizes, a normalised measure is easier to compare across pairs. To stay directly comparable to other methods such as TED or pq-grams, defining a distance is advantageous. Thus, our goal is to formulate a normalised distance from the absolute similarity.

In accordance of the identity profile projection operators based on sets and multisets, we adapt the well-known Jaccard distance. The Jaccard distance is defined on two sets A and B with

$$\mathrm{dist}^{\mathrm{Jaccard}}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}. \tag{4.22}$$

As desired, the Jaccard distance is a normalised distance, ranging from 0 if A and B are identical to 1 if the two sets share no common features. Based on this, we introduce a normalised distance for two trees T_1 and T_2 based on the concept of identity profile projections.

Based on Jaccard distance, we apply our formalisation of identity profiles given by $|T_1\rangle$ and $|T_2\rangle$ for trees T_1 and T_2 to calculate the normalised distance as the symmetric difference (compare Figure 4.12c on page 68):

$$\mathrm{dist}^{\mathrm{simple}}(T_1, T_2) = 1 - \frac{\langle T_1|1|T_2\rangle}{\langle T_1|1|T_1\rangle + \langle T_2|1|T_2\rangle - \langle T_1|1|T_2\rangle}. \tag{4.23}$$

This definition follows by reformulating Jaccard distance to replace set intersections with identity profile projections.

Proof. Let T_1 and T_2 be two trees and $V(T_1)$, $V(T_2)$ their respective set of vertices. Let $|T_1\rangle$ and $|T_2\rangle$ be the respective identity profiles based on the simple tree event stream representation of the given trees T_1 and T_2. The Jaccard distance can be expressed as

$$1 - \frac{|\,\mathrm{V}(T_1) \cap \mathrm{V}(T_2)|}{|\,\mathrm{V}(T_1) \cup \mathrm{V}(T_2)|} = 1 - \frac{|\,\mathrm{V}(T_1) \cap \mathrm{V}(T_2)|}{|\,\mathrm{V}(T_1)| + |\,\mathrm{V}(T_2)| - |\,\mathrm{V}(T_1) \cap \mathrm{V}(T_2)|}$$

Given that identity profile projections of two trees represent the overlap of identities, we can substitute $|\,\mathrm{V}(T_1) \cap \mathrm{V}(T_2)|$ by $\langle T_1|1|T_2\rangle$. This provides

$$= 1 - \frac{\langle T_1|1|T_2\rangle}{|\,\mathrm{V}(T_1)| + |\,\mathrm{V}(T_2)| - \langle T_1|1|T_2\rangle\rangle}.$$

Finally, we can substitute the size of a set of vertices $|\,\mathrm{V}(T)|$ by a identity profile projection of a tree onto itself, that is $\langle T|1|T\rangle$. This resolve to

$$= 1 - \frac{\langle T_1|1|T_2\rangle}{\langle T_1|1|T_1\rangle + \langle T_2|1|T_2\rangle - \langle T_1|1|T_2\rangle}.$$

□

Therefore, we define the absolute distance between trees T_1 and T_2 based on identity profile projection. Notably, the same definition is applicable to sets and multisets. In fact, it is valid for any well-defined identity profile projection operator.

Definition 4.33 (Insertion-only tree distance)**.** Let T_1 and T_2 be two trees, and their respective identity profiles $\langle T_1|$ and $|T_2\rangle$. The *insertion-only tree distance* $\mathrm{dist}^{\mathrm{simple}}(T_1, T_2)$ for the trees T_1 and T_2 is defined as the symmetric difference between their respective identity profiles.

$$\mathrm{dist}^{\mathrm{simple}}(T_1, T_2) = \langle T_1|\theta|T_1\rangle + \langle T_2|\theta|T_2\rangle - 2\langle T_1|\theta|T_2\rangle \tag{4.24}$$

Based on the insertion-only distance we can define the relative insertion-only distance. This matches the above derivation from the Jaccard distance, but is valid for all identity profile projection operators. A proof is omitted for brevity.

Definition 4.34 (Relative insertion-only tree distance)**.** The *relative insertion-only tree distance* between two trees T_1 and T_2 is derived from the insertion-only tree distance $\mathrm{dist}^{\mathrm{simple}}$, normalised by the union of the identity profiles of the two trees, that is

$$\mathrm{dist}^{\mathrm{simple}}(T_1, T_2) = 1 - \frac{\langle T_1|\theta|T_2\rangle}{\langle T_1|\theta|T_1\rangle + \langle T_2|\theta|T_2\rangle - \langle T_1|\theta|T_2\rangle}. \tag{4.25}$$

This definition generically covers a multitude of approaches, simply by the correct choice of identity class and identity profile projection operator. For example, the pq-grams distance is achieved with the identity class for dynamic pq-grams (see Section 4.4.4 on page 55) and the identity profile projection operator M.

Complexity

Space complexity for identity profile projection distances is proportional to the complexity of the underlying identity encoding strategy. This corresponds to a worst case of storing a fixed amount of data per identity. For fixed-length and infinite-length encoding, a sublinear complexity of $\mathcal{O}(n^{\bar{f}}), \bar{f} < 1$ is expected and a constant complexity $\mathcal{O}(1)$ can be achieved for a use case such as ours.

The given space complexity is partially limited by the underlying tree event stream representation. In the case of Simple tree event stream representation, only start events of vertices are available. As the tree traversal is arbitrary, vertices can potentially occur on every branch. Thus, it is not possible to identify and exclude finished branches, where future occurrences of vertices are impossible.

The total time complexity for tree distances depends on the size of the sequence of events to be processed. This, in turn, is given by the tree event stream representation for two trees T_1 and T_2 to be compared. All events must be converted to a collection of identities, and the conversion of each identity requires $\mathcal{O}(1)$ time. Thus, the conversion of all events has a time complexity of $\mathcal{O}(|S^{\text{simple}}(T_1)| + |S^{\text{simple}}(T_2)|)$. Consequently, the time complexity for identity profile generation is $\mathcal{O}(|S^{\text{simple}}|)$.

Each distance must determine the symmetric difference between two identity profiles. With identity profile projections based on hashing, we assume a time complexity of $\mathcal{O}(1)$ to check the existence of each identity for both trees. Thus, checking all identities for both trees symmetrically results in $\mathcal{O}(|S^{\text{simple}}(T_1)| + |S^{\text{simple}}(T_2)|)$ operations. Thus, the time complexity for identity profile projection is $\mathcal{O}(|S^{\text{simple}}|)$, as for identity generation.

The distance calculation for tree event stream representation requires two steps: the conversion of identities as well as the projection of identity profiles. Therefore, our overall time complexity for distance calculation is $\mathcal{O}(2|S^{\text{simple}}|) = \mathcal{O}(|S^{\text{simple}}|)$.

4.5.3. Dynamic Distance

Dynamic trees can be treated as a sequence of static trees (see Section 4.3.2 on page 48). Each intermediate state is a complete static tree that can be compared to other dynamic or static trees. This allows to use the same distance methods as for static trees.

Both definitions given in Equation (4.24) and (4.25) can also be applied to dynamic trees $\mathscr{T}_1$ and $\mathscr{T}_2$. Instead of the simple tree event stream representation S^{simple} we consider the dynamic tree event stream representation S^{dynamic} to determine the identity profiles of the given trees.

Definition 4.35 (Dynamic tree distance)**.** The *dynamic tree distance* $\text{dist}^{\text{dynamic}}$ is determined for the identity profiles based on dynamic tree event stream representation of two trees $\mathscr{T}_1$ and $\mathscr{T}_2$ at state i and j respectively, that is $\mathscr{T}_{1,i}$ and $\mathscr{T}_{2,j}$. The distance is given by

$$\text{dist}^{\text{dynamic}}(\mathscr{T}_{1,i}, \mathscr{T}_{2,j}) = \langle \mathscr{T}_{1,i}|M|\mathscr{T}_{1,i}\rangle + \langle \mathscr{T}_{2,j}|M|\mathscr{T}_{2,j}\rangle - 2\langle \mathscr{T}_{1,i}|M|\mathscr{T}_{2,j}\rangle, \tag{4.26}$$

with M denoting the multiplicity function.

Definition 4.36 (Relative dynamic tree distance)**.** The *relative dynamic tree distance* between two trees $\mathscr{T}_1$ and $\mathscr{T}_2$ is the result of dynamic tree distance $\text{dist}^{\text{dynamic}}(\mathscr{T}_1, \mathscr{T}_2)$ normalised with the union of the identity profiles of the two trees, that is

$$\text{dist}^{\text{dynamic}}(\mathscr{T}_{1,i}, \mathscr{T}_{2,j}) = 1 - \frac{\langle \mathscr{T}_{1,i}|M|\mathscr{T}_{2,j}\rangle}{\langle \mathscr{T}_{1,i}|M|\mathscr{T}_{1,i}\rangle + \langle \mathscr{T}_{2,j}|M|\mathscr{T}_{2,j}\rangle - \langle \mathscr{T}_{1,i}|M|\mathscr{T}_{2,j}\rangle}, \tag{4.27}$$

with M denoting the multiplicity function.

Complexity

Space complexity with regard to tree event stream representation decreases for dynamic tree distances compared to insertion-only tree distances. The explicit presence of exit events for vertices allows to reduce the size of the identity profile applicable at any time. For each vertex that is removed from the tree we can guarantee that no further events are assigned. Thus, while the tree event stream representation of dynamic trees is potentially bigger, at any specific point in time the same number of identities as for an equivalent static tree is sufficient.

Regarding time complexity for a single distance measurement there is no difference between insertion-only tree distances and dynamic tree distances. Both strictly depend on the number of events given by tree event stream representation. Thus, time complexity for dynamic tree distances for one single state is $\mathcal{O}(|S^{\text{simple}}|)$. However, if tree distance is calculated for each state i of a given dynamic tree $\mathscr{T}$, with $i = |\mathscr{T}|$ the complexity for time calculating all i distances is $\mathcal{O}(|S^{\text{simple}}|^2)$.

4.5.4. Incremental Tree Distances

Treating dynamic trees as a sequence of distinct static trees ignores the constraints of how each stage transforms to the next. For any non-trivial tree, the change between two stages is small compared to the size of the tree. Deriving the distance at each state separately implies that the distance calculation for most of the tree is repeated multiple times. Thus, the complexity for this approach is $\mathcal{O}(|\mathscr{T}|^2)$ (see Section 4.5.3).

An approach with this complexity does not scale well, especially true for streaming environments with limited resources. Therefore, an incremental distance function that reuses previous results is desirable for tree distance measurement. For this, it is sufficient to consider the gradual change of a dynamic tree between each stage.

An incremental distance for dynamic trees considers each event for distance calculation with respect to results from previous events. Thus, each step only checks for overlap of the current identity that is given by the specific event. For this, it is important to distinguish between a *recorded* tree $\langle\mathscr{T}_1|$ and a *currently observed* tree $|\mathscr{T}_2\rangle$ (see Section 4.5.1 on page 67). The $\langle\mathscr{T}_1|$ is a superposition of all intermediate, static states of $\mathscr{T}_1$. Only $|\mathscr{T}_2\rangle$ is a dynamic tree that changes over time as the tree event stream representation advances.

Definition 4.37 (Simple incremental dynamic tree distance)**.** The recurrence formula for *incremental dynamic tree distance* between a recorded identity profile of a recorded dynamic tree $\mathscr{T}_1$ and an observed dynamic tree $\mathscr{T}_2$ based on any tree event stream representation is given by

$$\begin{aligned} \text{dist}_0 &= \sum_j \alpha_j \langle\mathscr{T}_1|\theta_j|\mathscr{T}_1\rangle, \text{ with } \sum_j \alpha_j = 1 \\ \text{dist}_i &= \text{dist}_{i-1} - \sum_j \alpha_j (2\langle\mathscr{T}_1|\theta_j|P_{2,i}, Q_{2,i}, S_{2,i}, V_{2,i}\rangle - 1), \end{aligned} \tag{4.28}$$

where θ_j defines an identity profile projection operator to derive the distance component at each step, and α the weighting for these components.

A selection of distances for exemplary trees is shown in Figure 4.13 on the following page. The distance starts with a base distance which is the size of the identity profile of $\mathscr{T}_1$, that is $\langle\mathscr{T}_1|1|\mathscr{T}_1\rangle \leq |S^{\text{simple}}(\mathscr{T}_1)|$. This initialisation ensures a smooth distance

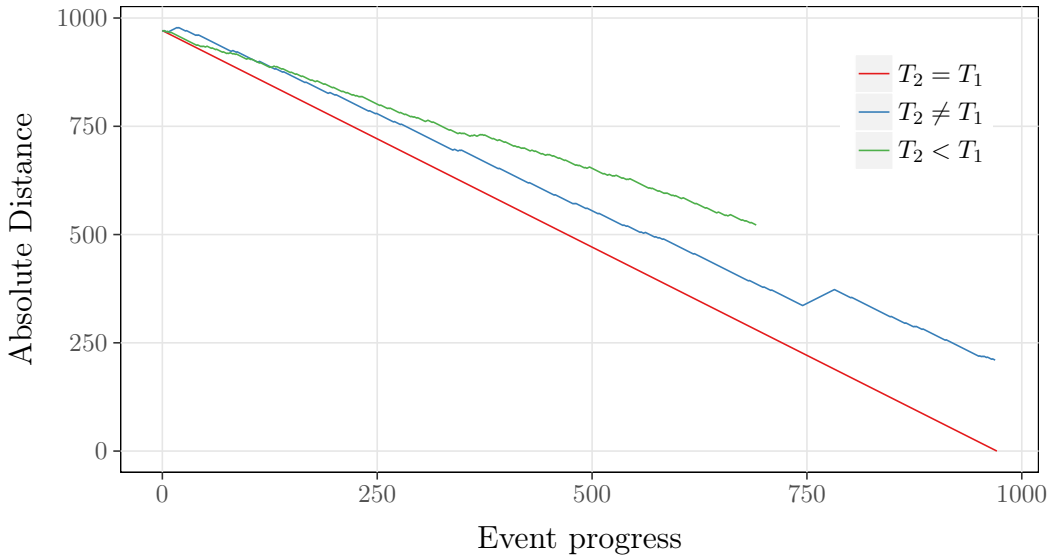


Figure 4.13.: Progression of incremental dynamic tree distance for the same tree T_1 against multiple variants T_2. The distance always begins at the maximum number of events in T_1. Comparing T_1 against the same tree T_2, where $T_2 = T_1$, means that each event matches and reduces the distance by one. Consequently the absolute distance reaches 0 after all events are processed. Comparing T_1 against a different tree T_2 of same size with $T_2 \neq T_1$ results in mismatches and matches. Mismatches increase distance by one while matches decrease distance by one, thus reaching a non-zero value in the end. For a smaller tree T_2, with $T_2 < T_1$, distance progresses normally but stops at the end of the event stream, remaining at the correct distance given missing events.

if the observed tree $\mathscr{T}_2$ has fewer vertices than $\mathscr{T}_1$. If the observed tree is prematurely finished, that is $|S^{\text{simple}}(\mathscr{T}_1)| > |S^{\text{simple}}(\mathscr{T}_2)|$, the distance is guaranteed to return a result $\text{dist}^{\text{simple}}(\mathscr{T}_1, \mathscr{T}_2) > 0$.

More importantly, this initialisation ensures that the distance over time replicates the distance to $\mathscr{T}_1$ at each step of $\mathscr{T}_2$. This mimics the convergence or divergence of $\mathscr{T}_2$ to $\mathscr{T}_1$: A tree that unfolds to a similar state to its observed counterpart decreases in distance, as additional matching vertices are revealed, eventually approaching a distance of 0. A tree that unfolds to a structure unlike its observed counterpart increases in distance, as additional vertices not part of $\mathscr{T}_1$ are revealed.

Notably, the difference between the current distance and the ideal distance at the current step is monotonic. Thus, we can reason about the progression of distance as a dynamic tree unfolds: an observed tree may stay the same or deviate from ideal behaviour, but it can never approach it after deviating from it. Thus, the progression of tree distance compared to ideal behaviour is bounded, allowing to draw conclusions before the entire tree is known. Due to the incremental design of distance function this analysis can be performed online, enabling near realtime responsiveness.

4.6. Summary

We have evaluated various tree distance approaches based on our requirements to process dynamic trees in a streaming environment (see Section 3.3 on page 35). Due to polynomial complexity of exact tree distance measures based on TED, we focused instead on three different classes to approximate tree distances: summary-based, decomposition-based, and time series-based approaches. Many of the analysed approaches are unsuitable for streams and dynamic trees. We identified decomposition-based approaches to be the most suitable concept. The methodology of decomposition itself follows an incremental approach. However, existing methods assume specific features for decomposition, such as traversal methods, preventing them from being generalisable for both dynamic trees and streams.

Based on the need for a generic, incremental, and extensible approach, we propose a framework for decomposition-based similarity and distance measures. Our approach focuses on modularity by separating the tasks of tree representation by decomposition and an appropriate distance measure. We express the decomposition as a profile of identities. Furthermore, we model similarity as identity profile projection of identity profiles or identities. This modular design allows future extensions and optimisations of individual tasks.

To support different classes of use cases, trees must be appropriately represented. We define four separate context dimensions to propose specific identity classes for tree representation. To meet the requirements of our use case, we restrict the dimensions by availability in the streaming model. Based on these definitions, we demonstrated how to integrate existing approaches from literature into the proposed framework to make them available for dynamic trees and stream processing. Furthermore we showed that space complexity of identity profiles is suitable for our targeted use case. In addition, the modularity of our approach allows to further improve this by sketching techniques for example.

From our formalisation, we naturally derived a distance measure for trees based on the projection of identity profiles and identities. We demonstrated this approach for static trees by succinctly replicating the definition of the common *pq*-grams. A naive application of this static distance for dynamic trees already shows complexity comparable to the complexity of TED for purely static trees. However, we have shown that our formulation allows for a conversion to a purely incremental calculation of distances. With this incremental distance measure, we preserve the sublinear complexity of our approach even for dynamic trees in a streaming environment. This generic dynamic tree distance is the basis for an extension to attribute-based distances.

5. Representation of Dynamic Tree Attributes

The formalisation of distances for dynamic streaming trees provides the framework to realise attribute-based distances. The evaluation of attributes in the context of tree distance measures adds further semantic meaning to the representation introduced previously. Within the scope of this Chapter we integrate the representation of attributes into our formalism to express semantics of attributes by their values. This compatibility with our proposed methodology enables an efficient and scalable analysis. A good approximation of attributes enables better discrimination of tree representation for further online analysis.

We further discuss two different strategies for representing attributes by summarising their values. The focus is on the provisioning of a stream-capable implementation of proper statistics. Challenges considered are different underlying types of attributes as well as their sparsity. Thus, we discuss strategies to properly represent granularity while enabling compression of representation as well as good time and space complexity.

5.1. Related Work

As outlined in Section 4.1 on page 39 there is a plethora of research to determine similarities and distances between tree-structured data. However, most of the research does not explicitly consider attributes except strings. In the following, we review the different available approaches to represent attributes for tree distance approximation. We supplement this review with a brief review for time series analysis as the analysis of dynamic trees results in time series of attribute values.

5.1.1. Attributes in Trees

Available approaches to tree distance approximation can be distinguished as content-based, structure-based, and hybrid methods utilising both content and structure. Most structure-based approaches utilise labelled, rooted trees. Thus, they usually consider the label of vertices to indicate siblings, children, ancestors, or the anchor vertex [49, 94, 149, 177]. These approaches rely on equality comparison based on labels. In general, explicit attribute values are disregarded when evaluating structural properties of semi-structured documents [73]. Content-based approaches focus on syntactic or semantic comparison of labels or text vertices [181, 216, 223].

Some approaches specifically encode attributes of vertices and their values: For example, Joshi et al. [123] append the name of attributes and their values in sorted order to a vertex' label. Augsten et al. [23] map vertices and their attributes to their respective label-value pair. However, these approaches do not consider the underlying type of attribute. Consequently, the alphabet of labels can become very large or even infinite given continuous data types. This approach renders most decomposition-based tree distances unfeasible as they rely on a finite alphabet for time and space requirements.

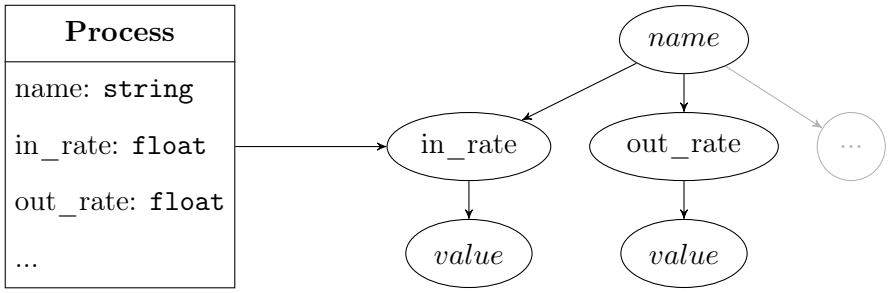


Figure 5.1.: Visualisation of mapping a process object with several attributes to hierarchical data [186, 236]. As an example, we consider one of the monitored processes within a batch job (see Section 3.2.4 on page 34 for details). The value of attribute *name* is interpreted as the label of the vertex. The key of each attribute, for example *in_rate* or *out_rate*, is added as a child vertex. Furthermore, each value is appended to its respective attribute vertex. Thus, for each attribute two additional vertices are added to the final tree.

Other approaches treat attributes as a part of the tree's vertices [167, 186, 236, 241]. Commonly, attribute vertices appear as children of their containing vertices sorted by attribute name, and appearing before all remaining vertices [167, 241]. Consequently, each attribute is decomposed into key and value. Both are represented as separate vertices, with the key and value interpreted as labels of their vertices. The attribute key vertex is the parent of the attribute value vertex. In the final tree event stream representation, each attribute vertex chain is a child of its owning vertex. Although we can differentiate vertices for elements and attributes the vast majority of existing approaches treat them equivalently [178]. Figure 5.1 visualises the approach for a vertex with two additional attributes besides its label.

This approach is consistent with the definition of TED. The interpretation of attribute key and attribute value ensures to evaluate similarity for two trees containing attributes. However, the underlying type of data is ignored or assumed to be uniform across the tree. For example, Ribeiro and Härder [186] only consider data of string type. Thus, approaches usually only differentiate the existence of attributes by applying 0 for mismatch and 1 for match of attributes. In addition, each attribute is represented by considering two additional vertices in the final tree. Thus, a missing attribute in one tree is penalised with an accumulated distance of 2. Existing distances for tree-structured data supporting attributes therefore tend to favour attributes over vertices.

The encoding of attributes as vertices within the tree makes their evaluation uniform with vertices. Several approaches from literature rely on fixed-length encoding for tree representation [22, 25, 233]. For example, in Augsten, Böhlen, and Gamper [22] the authors show that pq-gram distance is a good approximation to TED when p is fixed to 1. However, the constraint $p = 1$ implies that each attribute value is encoded only with its attribute key. Thus, attributes are processed independently from their owning vertex. Consequently, only the presence of attributes on vertices is guaranteed. In contrast, attribute values are summarised over the entire tree, without regard to hierarchy. This is a consequence of the diamond effect for fixed-length encodings (see Section 4.4.4 on page 58).

However, to the best of our knowledge, a flexible approach for differing attribute types in hierarchical data is missing. Several approaches propose solutions for the semantic analysis

of labels and text values only, based on results from information retrieval or linguistics. However, especially in the domain of dynamic trees, distances based on arbitrary attributes have not been considered so far.

Current research on tree distance approximation usually considers only one value per attribute. In the context of dynamic trees, values for attributes are subject to changes over time. Thus, we need to deal with time series of attribute values embedded into hierarchical data. In the following, we therefore briefly review methods for time series analysis suitable in our field of research.

5.1.2. Time Series Analysis

Time series analysis can be distinguished into instance-based and feature-based algorithms [129]. While instance-based methods are more exact, feature-based approaches gain efficiency by summarising data.

Instance-based methods usually focus on the shape of time series. The evaluation is based on similarity comparison of a test instance to a learning instance. In this field, nearest neighbour classifiers with Euclidean distance or Dynamic Time Warping (DTW) are considered the most successful and are widely used [117, 130, 224, 230]. While the Euclidean distance is efficient regarding space and time complexity, it is not robust to time-shifted data and noise, resulting in poor accuracy [183]. Both issues are generally handled by using elastic distance measures such as DTW. DTW [131] uses dynamic programming to determine the best alignment of time series yielding an optimal distance. As such, DTW is invariant to non-linear variations in the time dimension and considered a strong solution for time series analysis [184]. However, DTW has a time complexity of $\mathcal{O}(n^2)$ and is, therefore, not feasible in streaming environments.

Feature-based methods instead are generally more efficient regarding time complexity. These methods are based on characterising or summarising structural characteristics such as discontinuity [97] or statistical features [187, 226] of time series as a whole or in parts. Usually, feature-based methods extract local features at absolute points in time, assuming patterns to exist in the same time interval in different time series. This is problematic as patterns may be shifted in time across time series [97].

In the field of information retrieval, it is common to estimate word frequency ignoring spatial characteristics by utilising the bag-of-words principle [159]. Recently, this approach has been adopted for time series classification. The idea is to estimate the multiplicity of local characteristics of the time series and then use these measures as new features for a classifier [32, 152, 153].

In Lin, Khade, and Li [152] and Lin and Li [153] the authors utilise a sliding window approach. They adapt the symbolic representation for time series called Symbolic Aggregate approXimation (SAX) [154]. SAX converts the original time series data to a lower dimensional representation of symbolic words. These words represent patterns in the time series. The incremental construction of this time series representation into a histogram renders this approach feasible for streaming environments. In Baydogan, Runger, and Tuv [32] the authors consider fixed and variable-length intervals to extract multiple features from time series. However, this requires a pre-processing, standardising time series to zero mean and unit standard deviation to adjust different baselines and scales.

Both approaches have shown to be superior to existing approaches in terms of clustering, classification, and anomaly detection. Furthermore, the concept to decompose a time series in order to perform further analysis on a representation by multisets is in line with our

formalisation introduced in Chapter 4 on page 39. Thus, we expect such an approach to integrate well with our own. Still, both approaches cannot be directly applied to streams. For example, deriving the mapping to logical words for SAX requires knowledge on the distribution of values. In addition, both methods require the buffering of values. Ideally, we desire to minimise the required buffering of values. The main reason for this strict limitation is that attributes are only one aspect of dynamic tree distance measurement and therefore must not dominate space or time complexity.

5.2. Encoding Attributes

Analysis of related work revealed that most solutions to tree distance approximation do not explicitly consider attributes. Instead, we extend our existing approach to tree decomposition to also include time series of attributes. This has shown to be a reliable approach for time series data analysis (see Section 5.1.2 on the preceding page).

To enable the handling of attributes, we integrate attributes into our formalisation of dynamic trees. In the following, we extend the current definitions of identities, identity profiles, and distances from the current formalisation (see Chapter 4 on page 39). Notably, the goal is to generalise existing definitions with extensions for attributes. The extended formalisation remains valid for trees without attributes.

Each tree represents a hierarchical relationship between the vertices it contains. In turn, attributes of a vertex can be seen as part of the vertex' hierarchy. However, it is important to distinguish between these two hierarchies: The tree itself represents a hierarchy of objects. Each object represents its attributes as a flat hierarchy. To better express these nested hierarchies, in the following we refer to vertices that represent objects with attributes as *composite vertices*.

Definition 5.1 (Composite vertex)**.** Let T be a tree with a set of vertices $\mathrm{V}(T)$. Each vertex $v \in \mathrm{V}(T)$ representing an object with a set of attributes A, with $|A| \geq 1$, is called a *composite vertex*. The label of a composite vertex is given by a specific attribute A_i that identifies the object. The parent of a composite vertex is another composite vertex, if any. A composite vertex can have any number of children.

5.2.1. Naive Encoding of Attributes

To motivate our approach for attribute handling in dynamic trees, we first introduce a naive approach. The technique is similar in parts to the handling of attributes in common tree distances (see Section 5.1 on page 77). Instead of treating attributes as separate *features* of vertices, we treat them as *defining* feature of vertices. This mimics how we have implicitly used vertex labels, which are also attributes, to identify vertices. This serves to demonstrate the limitations of a pure identity and cardinality model, and the requirements to treat attributes separately of identity.

Let T be a static labelled tree where each vertex $v \in \mathrm{V}(T)$ is a composite vertex with several associated attributes A, in addition to its label. Its respective tree event stream representation is $S^{dynamic}$, though the end events are ignored for simplicity. Furthermore, we assume that each attribute A has the same underlying type of data from the domain of values $\{1, \ldots, k\}$ for some finite, but possibly large k.

A naive approach to treat attributes is to include specific attribute values into a vertex' identity. We introduce another dimension to reflect characteristics of attributes:

- The *attribute* $\mathcal{A}(v)$ associated with a vertex v in the tree reflects the local characteristics of the vertex.

Thus, a naive attribute-supporting identity can be defined as $|\mathcal{P}, \mathcal{Q}, \mathcal{S}, \mathcal{V}, \mathcal{A}\rangle$, where $\mathcal{A}$ specifies a given value of attributes A. Notably, this identity does not require to extend the tree as proposed in literature (see Section 5.1.1 on page 77). Hence, the number of identities that are computed for a identity profile does not increase with this approach. Furthermore it has the advantage that attribute evaluation uses the same algorithm.

This naive approach can be easily applied to the formalisation presented in Chapter 4 on page 39 as it does not require to change the definition of identity profiles and distances based on identity profile projections. However, it implies several disadvantages relating to a) the number of distinct attributes and values, b) the unfolding of attribute values.

Number of distinct attributes and values If the attribute value is part of the identity, the alphabet size of the identity is proportional to the alphabet size of any given attribute. Thus, the size of the identity profile does not only depend on the number of vertices but also on the number of distinct values of attributes associated to vertices. For categorial data this is bounded, but in case of continuous data it can become infinite.

This makes it unlikely that multiple vertices share the same identity. Even a small change of any attribute creates a distinct identity. In turn, it is not feasible to match vertices based on their attribute-supporting identity. Instead an explicit search for similar vertices is required. Consequently, time complexity per vertex is $\mathcal{O}(n)$, instead of $\mathcal{O}(1)$ without attributes.

Furthermore, space complexity depends on Property 4.4 on page 50, which relies on a finite alphabet of identities. The lack of shared identities for vertices means compression from collisions is unlikely. Therefore, one cannot expect sublinear space complexity.

Consequently, any approach to include attributes into tree representation that targets applicability in streaming environments as well as scalability requires a method for discretisation, aggregation, or sketching of attribute values.

Influence of sparsity of attributes Given a dynamic tree $\mathscr{T}$, the unfolding of the tree together with attribute values increases the number of identities. Each event only describes a single edit operation within the tree (compare Section 4.4.3 on page 51). Therefore, it only describes a single change of the value of one attribute A_i. In turn, the values for attributes $\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m\}$ are unknown for the current event. Thus, the generated identities become sparse in attribute dimension.

To prevent this sparsity, values for each attribute A_i per vertex require buffering. Notably, this must be performed before compressing a vertex to its identity. This can arbitrarily delay the processing of a vertex waiting for additional attribute values.

In addition to the scope of a single object, sparsity also affects the available attributes for all other objects. Not every object modelled as a vertex $v \in \mathrm{V}(\mathscr{T})$ requires the same set of attributes $\{A_1, \ldots, A_m\}$. Thus, the identity class needs to encode each of the available attributes $\{A_1, \ldots, A_{m_i}\}, \forall v_i \in \mathrm{V}(\mathscr{T})$. This, in turn, raises space complexity to $\mathcal{O}(nm)$. Even if few vertices have attributes, space complexity depends on the total number of distinct attributes in the entire tree.

Implications for the Encoding of Attributes

The naive approach to directly encode attributes into the identity of a vertex is not feasible due to scalability and complexity constraints for dynamic trees in streaming environments. Direct encoding of attribute values prevents identity matching. Treating attribute categories as separate dimensions of identities increases space complexity. Thus, attributes must be treated orthogonal to identities.

Instead, we adapt the approach to encode relevant attributes within the tree. To meet the requirements of dynamic trees in streaming environments we impose the following requirements:

1. efficient handling of *distinct* attributes and values,
2. efficient handling of *sparse* attributes and values,
3. compatibility with different types of attributes including labels, and
4. continuous mapping of value similarity to distances instead of binary matching.

5.2.2. Attribute-Supporting Tree Model

So far, we have defined the identity profile projection of identities to yield either zero or maximum similarity. This reflects that identities match comparable vertices, which is either possible or not. In contrast, we desire a continuous range for distances based on attributes, which themselves can have continuous values. Therefore, we need to separate the identity matching of vertices, attribute type, and the comparison of attribute values.

To encode relevant attributes of vertices in a tree we partially adopt the method proposed in literature to express attributes as vertices. Thus, each attribute that is related to a specific composite vertex is represented by a new vertex. However, we explicitly distinguish these attribute vertices from previously defined vertices. Attributes define an attached atomic value and are in the following restricted to not contain references to further composite vertices. For better distinction, we introduce the concept of *atomic vertices* to represent attributes.

Definition 5.2 (Atomic vertex)**.** Let v be a composite vertex and A_i an associated attribute. The *atomic vertex* is a vertex representing the associated attribute A_i of v. An atomic vertex is labelled with the key of the specific attribute A_i. The parent of an atomic vertex is the respective composite vertex v. An atomic vertex never has children by itself.

Note that the label of a composite vertex is restricted to an attribute that statically identifies the related object. By utilising a static identifying attribute to denote the composite vertex, we ensure minimal distance distortion from attribute changes within a dynamic tree.

Based on the definition of composite vertices and atomic vertices we define the attribute-extended tree.

Definition 5.3 (Attribute-extended tree)**.** Let T be a tree, and a_i the number of attributes for the composite vertices $v_i \in \mathrm{V}(T)$, that is $a_i = |\{A_1, \ldots, A_{m_i}\}_{v_i}|$. The *attribute-extended tree* is constructed from T by adding $a_i - 1$ atomic vertices to each composite vertex $v_i \in \mathrm{V}(T)$.

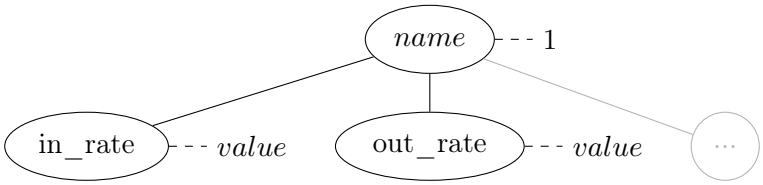


Figure 5.2.: Visualisation of the concept to map dynamic attributes to trees. The visualisation shows one composite vertex that is labelled *name*. This label is the value of one selected attribute A_i. As the value of attribute A_i is utilised as the label of the composite vertex, a weight of 1 is assigned as a default. The remaining $a_i - 1$ attributes are appended to the composite vertex as atomic vertices. Each atomic vertex is labelled with the name of its attribute A_j. Furthermore, the value of attribute A_j is assigned as a weight.

Unless otherwise stated, we base the following discussions on the definition of attribute-extended trees to define distances for dynamic trees. Thus, trees explicitly include attributes in the following.

However, in contrast to approaches from literature, we do not add vertices for the *values* of the given attributes. First, this unbalances the distance calculation as each attribute except the label is counted twice (attribute key and attribute value). Second, the label of the composite vertex is comparable to the attributes modelled by atomic vertices. Thus, they need to be treated equally for distance calculation. This is not the case when also integrating attribute values as vertices.

Instead, we model the attribute values as attributes to the tree's vertices. Therefore, we adapt the definition of weighted, labelled trees. We introduce an *attributed, labelled tree* T as $T = \{V, E, A\}$. Extending the rooted tree introduced in Section 4.3.1 on page 47, the attributed, labelled tree also consists of a finite set of attribute values $\mathrm{A}(T)$. Each vertex that has no explicit assigned attribute value, for example composite vertices, defaults to an implicit value of 1.

We explicitly do not replicate the original definition of weighted, labelled trees here, as this limits weights to numbers [66]. To properly represent attributes, we require a variety of data types, such as numeric types, string types, boolean types, or categorial types. Figure 5.2 visualises the process object from Figure 5.1 on page 78 based on our definition of attribute-extended trees. Values are not included in hierarchical tree information but give a semantic context to vertices.

5.2.3. Attribute Identities

Our approach for expressing attributes distinguishes itself from approaches in literature by the use of explicit, different types of vertices. The composite vertices represent labelled objects, which logically correspond to regular vertices of classical approaches. In contrast, each atomic vertex represents features of these objects, as both an attribute label and value. However, to benefit from this differentiation requires special treatment of each type.

The identity of a vertex must distinguish between composite vertex and atomic vertex to properly reflect its type. Most importantly, the dimensions defining the local context (see Section 4.4.3 on page 54) are not meaningful for atomic vertices. Still, it is important for us not to conflict with the previous handling of vertices.

To preserve the previous definitions, attributes must not conflict with the existing formal-

isation. Rather, the attribute-extended tree should be compatible with the formalisation presented in Chapter 4 on page 39. In fact, our previous choice of distances implicitly expresses an attribute-based distance measure for composite vertices with one attribute, namely its label.

Proof. We consider an attributed, labelled tree T with a set of composite vertices $\mathrm{V}(T)$. For comparability, we assume that each composite vertex $v \in \mathrm{V}(T)$ has a single associated attribute A_l. The value of A_l is 1, and the key is the label of the composite vertex. Let $|\mathcal{P}(v) = \varnothing, \mathcal{Q}(v) = \varnothing, \mathcal{S}(v) = \varnothing, \mathcal{V}(v)\rangle$ with $\mathcal{V}(v) = |v.lbl\rangle$ the identity class. As the value for each composite vertex is 1 all identities associate the same property 1. Thus, the count of each identity and the sum of all its values are equal. Depending on the specified identity profile projection operator θ, such as 1 or M, we derive a distance based on existence of the label of the given attribute A_l or its multiplicity by counting it or summing its values. □

The distinction of vertex types is motivated by the assumption that comparing attributes of different objects is never meaningful. In other words, atomic vertices of a composite vertex are only meaningful when compared to atomic vertices of similar composite vertex. As the identity already expresses similarity, we restrict attribute comparison to composite vertices with matching identity.

Example 5.4. Consider a process downloading data from a server, and another performing calculations on this data. For each process, the total network traffic and processing time is recorded. The chain of both processes is repeated multiple times and the repetitions are compared.

It follows from this explanation that it is not meaningful to compare network traffic of a download process with a calculation process. They are expected to differ. However, traffic is not sufficient for a precise identification of each process. Traffic is expected to differ slightly, for example if network packets are lost. Still, we can compare traffic directly between processes of the same kind.

Consequently, the $\mathcal{V}$ dimension of vertices must reflect the associated composite vertex. In case of composite vertices the vertex under consideration can be directly encoded into the identity. In case of atomic vertices, the parent composite vertex is required in conjunction with the atomic vertex.

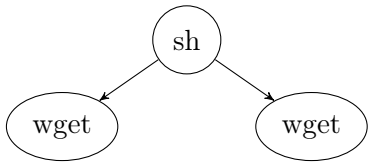
This grouping of atomic vertices and composite vertex to a canonical entity is deemed as the simplest semantically meaningful structural entity [216]; only considering the atomic vertex without context is otherwise meaningless [207, 208]. However, we notably use only the identity of each vertex, not any associated value.

Thus, we encode in $\mathcal{V}$ dimension

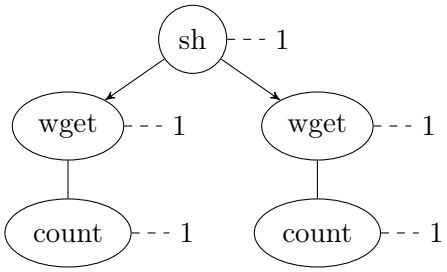
$$\mathcal{V}(v) = |v.compositenode.lbl, v.lbl\rangle, \tag{5.1}$$

where we again assume static labels of vertices for simplicity. This encoding enables a unified treatment of composite vertices as well as atomic vertices. Consequently, our approach realises a unified distance measurement for dynamic trees that includes attributes.

Proof. So far, we considered an identity class defined as $\mathcal{V}(v) = |v.lbl\rangle$ relating to the label of a vertex. Figure 5.3a on the facing page visualises the concept considered so far for one specific composite vertex. Based on this representation the identity profile projection operator M is used to determine the multiplicity of a given identity. In the following, we



(a) Consideration of multiplicity of vertices in a tree without specific attribute handling.



(b) Proposed approach to handling of attributes in trees

Figure 5.3.: Comparison of a tree with and without specific attribute handling. Figure 5.3a on the left visualises the approach considered so far without handling of vertices. The tree visualised on the right in Figure 5.3b shows the proposed approach to attribute handling in trees with an assigned attribute *count*. For demonstration purposes the value of count is 1.

introduce a further attribute *count* with a fixed value of 1 to each given vertex in the tree T. Thus, each composite vertex has an assigned atomic vertex for attribute count with value 1. This extension to the composite vertex is shown in Figure 5.3b.

We generate the identities for composite vertices as well as atomic vertices describing the attribute count. By applying the multiplicity identity profile projection function we can show that the following is valid:

$$M|wget\rangle = M|wget, \varnothing\rangle.$$

This follows by extension of $\mathcal{V}(v) = |v.lbl\rangle$ to $\mathcal{V}'(v) = |v.lbl, \varnothing\rangle$. The addition of a constant $\varnothing$ does not add further information, resulting in equivalent identities for $\mathcal{V}$ and $\mathcal{V}'$. Consequently, the attribute carries the same information for the actual cardinality of identity as the composite vertex. Obviously, counting attributes of value 1 and summing over their values is equivalent. Thus, the multiplicity of atomic vertices and the sum of their attribute *count* are interchangeable. Given a value summation operator M', we derive

$$\begin{aligned} M'|wget, count\rangle &= M|wget, count\rangle \\ &= M|wget, \varnothing\rangle \\ &= M|wget\rangle. \end{aligned}$$

□

In summary, the extended attribute-supporting encoding of identities has several advantages: It is beneficial for space complexity and compression rates. Compared to approaches from literature, values are not included in identities. Thus, only distinct attribute *keys* per composite vertex have an impact on required space. More importantly, the distinction between attribute key and value enables evaluation of existence of an attribute independent from its value. Consequently, distance measures can apply specific weighting of either part. Most importantly, it enables value-specific measures. These measures are not restricted to results of either zero or maximum distance but allow for continuous ranges.

For consistency, we following consider the shorthand notation $|v.lbl\rangle$ for the collection of identities regarding a given composite vertex $v \in \mathrm{V}(T)$. The shorthand notation is defined as

$$|v.lbl\rangle = |v.lbl, \varnothing\rangle + |v.lbl, a_1.lbl\rangle + \ldots + |v.lbl, a_k.lbl\rangle, \forall a_i \in A, \tag{5.2}$$

where A is the set of attributes, excluding the attribute identifying the composite vertex. This shorthand notation also ensures symmetry with our proposed formalisation.

5.3. Attributed Tree Distances

We define the integration of attributes by introducing the differentiation between composite vertices and atomic vertices. Based on this differentiation we have derived the formal expression of attributes that generalises attribute type and value.

We further require an adapted distance function to leverage this formalisation. So far we introduced the concept to distance measurement by replicating the evaluation of the multiplicity identity profile projection operator by considering the new concept of attributes. In the following we formally introduce this concept with respect to different attribute types and statistics.

5.3.1. Measuring Attribute Values

By introducing attribute identities, both vertices and attributes support identity profile projection and thus distance calculation. Consequently, Equation (5.2) allows to express the distance of composite vertices via their attributes. Thus, the calculation of $\langle v.lbl|M|v'.lbl\rangle$ results in valid *global distance* for a composite vertex v (compare Figure 5.3 on the preceding page) and its associated atomic vertices.

Global and Local Attributes

It is important to note that the global distance $\langle wget|M|wget'\rangle$ is only meaningful between trees, and not individual vertices. The distance is based on the identity profile projection operator M. This identity profile projection operator derives the multiplicity of identities in a tree. As this operation relies on a collection of identities, we refer to this measure as a *global attribute*.

Definition 5.5 (Global attribute)**.** A *global attribute* is a statistic based on the aggregation of values from a collection of identities.

A global attribute is not meaningful with only a single vertex or identity, but requires knowledge about the tree up to a given state. For example, the identity profile projection operator M introduced in Section 4.5.1 on page 69 derives the global attribute multiplicity for each distinct identity from a collection of identities. In this case the global attribute is a simple count or sum of distinct values. Still, the aggregation function deriving a global attribute can in principle be arbitrarily complex.

More naturally, we also describe the concept of *local attributes*.

Definition 5.6 (Local attribute)**.** A *local attribute* is a statistic derived from a single value of a given identity.

Local attributes do not depend on information of other vertices or the tree as a whole. The most natural representation of a local attribute is the value of an attribute itself without any conversions. Similar to how M maps to a global attribute, in Section 4.5.1 on page 69 we also implicitly introduced an identity profile projection operator that derives a local attribute: the identity profile projection operator 1. For each value or label the function determines whether the given identity exists in a recorded identity profile.

Representing Attribute Values

As shown when introducing attributes, global attributes such as vertex multiplicity can be expressed both as counts and sums of trivial attributes. Yet another view is to represent the multiplicity of identities by a histogram. This histogram only requires one bin, namely that of the single value all attributes share. Consequently, the global attribute on multiplicity for a given value is the volume of this bin.

Figure 5.4a on the next page visualises the histogram of the identity for the composite vertex wget in the tree given in Figure 5.3b on page 85. For each value 1 that is assigned to the composite vertices the count of the bin at position 1 in the histogram is incremented by 1 accordingly. Thus, distance calculation results based on multiplicity function M can be expressed as the difference of the value bin.

Local attributes can be distinguished from global attributes in that a single attribute is meaningful. Not only is the aggregation of attributes of relevance, but also individual values. This can be expressed again by a distribution, but over multiple values.

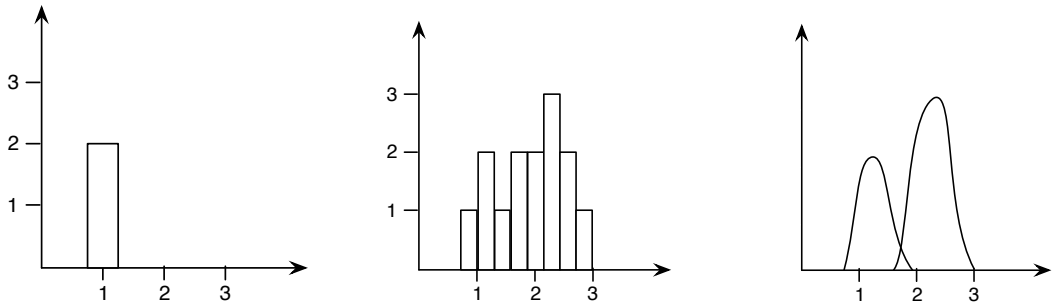
Thus, we can express local and global attributes with the same mechanism: The values of any attribute form a distribution of discrete, continuous, or categorial values. Global attributes are an extreme case, where values are not relevant.

Distributions again lend themselves to the simile of identity profile projections. The distance between attributes can be expressed via the overlap and difference of distributions, as visualised in Figure 5.4 on the next page. Notably, this generalises the notion of multiplicity distance, similar to how attributes generalise counts.

In specific, Figure 5.4b on the following page visualises the histogram representing multiplicity of identities for atomic vertices for the composite vertex wget.

Each of the attribute values representations in Figures 5.4a to 5.4b on the next page allow for direct distance measurement. In Figures 5.4d to 5.4e on the following page the statistics for a similar composite vertex are depicted. As for distance measurement based on identity profile projection, the distance can be described by symmetric difference. In Figures 5.4g to 5.4h on the next page the overlap of each of the histograms is visualised. Overlapping areas are depicted with a green colour, whereas remaining areas are depicted with red colour.

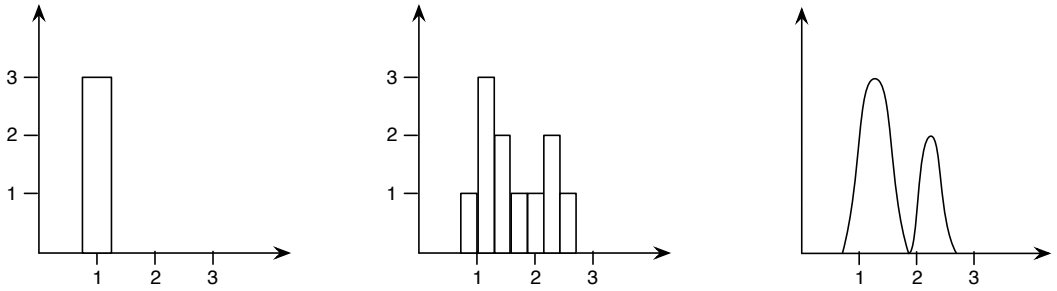
By design, histograms are restricted to a fixed granularity. Continuous ranges of distance values require the representation in terms of PDFs. Figure 5.4c on the following page visualises the PDF for the histogram of the adjacent Figure. Again, we can determine the distance by considering symmetric differences – the technique of deriving distances does not depend on the type of distribution. In contrast to histograms, a PDF allows for a continuous overlap and thus continuous distance results as visualised in Figure 5.4i on the next page.



(a) Global attribute of a vertex $v \in T$ with regard to identity profile projection operator $M|v.lbl\rangle$.

(b) Distribution of local attributes of vertices $v \in T$ based on a histogram.

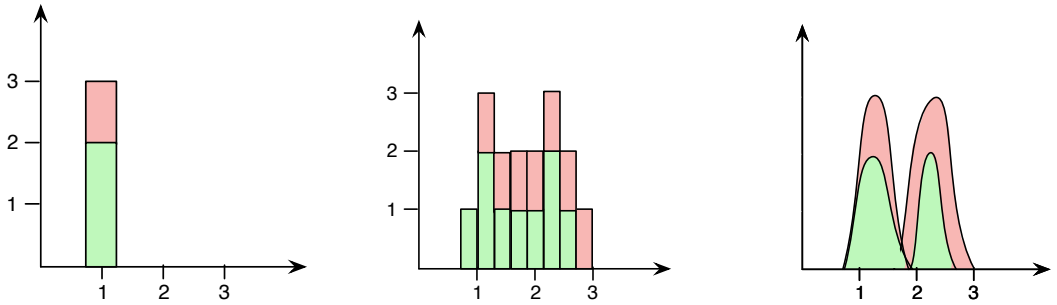
(c) Distribution of local attributes of vertices $v \in T$ based on a probability density function (PDF).



(d) Global attribute of a vertex $u \in T'$ with regard to identity profile projection operator $M|u.lbl\rangle$.

(e) Distribution of local attributes of vertices $u \in T'$ based on a histogram.

(f) Distribution of local attributes of vertices $u \in T'$ based on a PDF.



(g) Identity profile projection $\langle v.lbl|M|u.lbl\rangle$.

(h) Identity profile projection of two histograms for trees T and T'.

(i) Identity profile projection of two PDFs for trees T and T'.

Figure 5.4.: Effect of local and global identity profile projection operators with regard to underlying attribute statistics representation. Each of the first and second row of Figures visualises different exemplary statistics for a group of vertices having the same identity. The first column visualises the identity profile projection operator M to determine the global attribute multiplicity. The second and third column visualise two multiplicity identity profile projection operators that are defined on local attributes. The second column visualises the multiplicity based on a histogram whereas the third visualises the multiplicity by utilising a PDF. Finally the last row of Figures visualises the identity profile projection regarding the three different identity profile projection operators.

Approximating Attribute Values

Expressing multiple attributes with a single distribution is similar to how multiple vertices are expressed with a single identity. It is likewise a compression of a time series of data (see Section 5.1.2 on page 79). Yet, distributions must be suitable for stream processing, similar to constraints on identities.

Space complexity of a binning approach depends on the number of bins that are created from the underlying data. This is efficient when a single bin holds most data, but has to trade granularity for space complexity if distributions are smeared out. As such, histograms are favourable if the required granularity is coarser than the spread of data.

The representation of data matching a specific distribution with a corresponding PDF requires less space, aside from trivial distributions. For example, compare the two Figures 5.4b to 5.4c on the preceding page. Figure 5.4b on the facing page visualises a Gaussian distribution based on a binning to the integer domain whereas Figure 5.4c on the preceding page visualises the distribution itself. This distribution can be represented by three variables: the mean μ, standard deviation σ, and the number of samples. Storing three values per distribution is superior in space complexity to any non-trivial histogram.

However, not only values must be described efficiently. By definition, our approach to tree distance measures utilises identities to represent the vertices of a given tree. This results in multiple vertices sharing the same identity. The identity itself does not include attribute values. As we expect the range of attribute values to be sparsely populated, subgroups of vertices with the same identity can differ in this regard. For an attribute value distinguishing subgroups of a given identity, we therefore expect a finite number of distinct groups in the distribution of attribute values.

Example 5.7. Consider a shell script downloading several configuration files and executables. While the file sizes of configuration files and executable files differ, the file sizes in each group are roughly the same. For each file, a *wget* process is spawned. As each process has the same parent, the identity $\mathrm{Id}^{\mathrm{P}} = |\mathrm{wget}, \mathrm{bash}, \ldots\rangle$ is the same for each process.

However, the recorded traffic for each process is an indicator for the size of the downloaded file. The processes of each file group create a group of similar traffic values. Thus, the distribution of traffic for the identity $|\mathrm{wget}, \mathrm{bash}, \ldots\rangle$ exhibits two distinct groups.

Deriving the properties of the overall population requires the buffering of a number of data points from the population. A unimodal distribution does not fit groups of sub-populations and would therefore result in distortion of distance calculation. Thus, we strive to describe the overall, sparse population by the different sub-populations. This, in turn, enables the minimisation of required memory for distance calculation.

This problem is known in literature as a mixture model [51, 55, 238]. Mixture models are used to derive the properties of sub-populations given only observations of the overall population without prior knowledge about sub-populations. In the following, we do not consider the generalised approach but focus on Gaussian Mixture Models (GMMs). Consequently, we restrict ourselves to the assumption that underlying sub-populations are normally distributed. This assumption matches the constraints for most available machine learning and data analysis methods based on the central limit theorem [27, 29].

Gaussian Mixture Models In GMMs it is assumed, that all data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. The GMM is the weighted sum of Gaussian component densities. The parameters are estimated

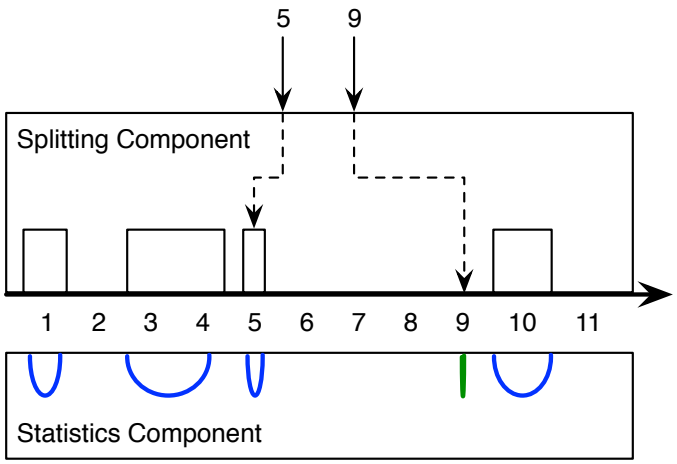


Figure 5.5.: Overview of the SplittedStatistics approach. The concept builds on two components: a *splitting component* and a *statistics component*. While the splitting component manages the creation and merging of bins by assigning appropriate values, the statistics component handles the representation of values by aggregating relevant statistics. The original values that are inserted into the statistics component are rejected after processing.

from training data by utilising the iterative Expectation-Maximisation algorithm [206]. Incremental GMMs have a time complexity of $\mathcal{O}(nkd^2)$ for n data points, k Gaussian components, and d dimensions.

However, the analysis of distribution of attribute values based on GMM is not appropriate given the constraints of online analysis of dynamic trees. Especially space complexity does not meet our requirements as all components need to be remembered. As an alternative, we propose an approximating approach that is based on ideas from GMM but focuses on efficiency and scalability.

SplittedStatistics We propose a dynamic approach, the *SplittedStatistics*, that approximates the learning of underlying distributions by utilising an incremental clustering of attribute values. The approach automatically adapts to the required sensitivity of the considered attribute.

The SplittedStatistics approach is based on two ideas: Different vertices can be represented by a shared identity, with groups of vertices adhering to different attribute value distributions. The first step targets the splitting of the underlying distributions. The second step focuses on the representation of distributions. This targets the minimisation of data points required to represent the actual distribution.

We therefore propose the SplittedStatistics consisting of two components: a) the *splitting component* and b) the *statistics component*. Figure 5.5 visualises the interaction between the two components. Again, this two-step procedure focuses on modularity and allows independent optimisation and configuration for both components. In the following, we first introduce the two components building upon one another. Based on this concept, we introduce and evaluate two methods to realise attribute-supporting tree distances.

Algorithm 2 Incremental assignment of data to dynamic bins

Precondition:
`x` is any value that should be assigned to the current sorted list of `bins`
λ is the distance threshold to determine the association to a bin
Postcondition: Updated list of `bins`

```
1:  function ADD(x)
2:      if |bins| > 0 then
3:          index ← index of closest bin to x
4:          if distance(bins[index], x) > λ then
5:              bin ← new bin(x)
6:              insert bin in order into bins
7:          else
8:              bins[index] ← bins[index] + x
9:              MERGE(bins, index)
10:     else
11:         bin ← new bin(x)
12:         append bin to bins
```

Splitting Component The splitting component is based on splitting values into groups to finally describe each group statistically. The splitting component defines a dynamic clustering of values into bins. The approach is shown in Algorithm 2. For each new value x of a given attribute, the distance to the closest existing bin is determined. If the distance is below a predefined threshold λ, the value is inserted into the bin. Otherwise, if there are no existing bins or no bin is close enough, a new bin is created and the value inserted. The handling of the insertion of values into a specific bin is managed by a statistics component. The statistics component defines the underlying statistic in each bin, such as a histogram or PDF.

After adding a new value to a bin, the splitting component checks if neighbouring bins can be merged. The merging approach is shown in Algorithm 3 on the following page. This merging procedure depends on the overlap of the statistics instances of two neighbouring bins. By merging the statistics of two bins, the component ensures space-efficient representation of data.

The statistics themselves are independent from the splitting. This distinction allows to individually react to specific events, such as low memory. If required, the available size of the data structure, that is the available number of bins, can be reduced, resulting in a greater approximation and mixing of statistics. Constraining available memory results in a loss of precision but ensures availability of distance results.

Statistics Component To represent the statistics for the attribute under consideration, different statistical approaches can be used. One approach that is commonly used in literature, is the consideration of the count [123, 126, 153]. However, it has been shown that the representation with PDF results in a higher precision [119]. Still, arbitrary distributions cannot be matched efficiently by a single PDF. Thus, we abstract from the statistic in use by introducing a generic statistic component. This ensures flexibility for different use cases.

The statistics component must summarise a stream of values $X = (x_0, \ldots, x_n)$ that are

Algorithm 3 Incremental merging of dynamic bins

Precondition:
`index` is the index of a bin within the sorted list of `bins` that is checked for merging
ω is the required distance threshold for two bins
Postcondition: Merged list of `bins`

```
1: function MERGE(bins, index)
2:     repeat
3:         merged ← False
4:         if distance(bins[index], bins[index + 1]) < ω then
5:             bins[index] ← bins[index] + bins[index + 1]
6:             remove bins[index + 1] from bins
7:             merged ← True
8:         if distance(bins[index − 1], bins[index]) < ω then
9:             bins[index − 1] ← bins[index − 1] + bins[index]
10:            remove bins[index] from bins
11:            index ← index − 1
12:            merged ← True
13:    until merged = False
```

assigned to a specific bin. Similar constraints apply as for identities: Space complexity must be sublinear to the number of values n, and ideally linear to the number of identities. Additionally, only a limited number of values may be buffered. The main focus is to retain a high precision while storing only a small amount of values X' with $|X'| << |X|$.

Generalising the utilisation of global and local attributes, we introduce two different statistic component approaches for the concept of SplittedStatistics. The first is a generalised approximation of Gaussian Mixture Model to smoothly model the population of an identity as an *incremental PDF statistics*. However, as the incremental PDF statistics makes a strong assumption on Gaussian distributions, we also introduce a *MultisetStatistics* to robustly model arbitrary distributions without prior knowledge.

Incremental PDF Statistics The learning of precise PDFs in a streaming environment is not efficient considering time and space complexity. We therefore consider a Gaussian distribution to represent the original data. To realise an approximating GMM we require the statistics component to represent a Gaussian distribution. Therefore, it calculates a *running mean* as well as a *running variance* to summarise given input values. To focus on scalability regarding time and space complexity, we process each value once to update mean and variance before dropping it completely. Thus, we do not require to buffer any values, but solely rely on the storage of running mean, variance, and the number of samples.

This method is advantageous for space complexity. By only considering the number of samples, running mean, and running variance per distribution, space complexity only depends on the number of represented distributions k, that is $\mathcal{O}(k)$. This, in turn, means, that further details are lost. Once, a distribution is merged with another distribution, it cannot be recovered. Furthermore, this makes the approach dependent on the order of values being added. These effects can be limited by considering the buffering of a limited number of values within each distribution.

Another disadvantage is the limitation to Gaussian distributions. However, the flexibility of the SplittedStatistics approach enables differing statistics calculations. To represent any underlying distribution instead of being limited to Gaussian distributions, an extension may consider the calculation of running quantiles. This does not limit the approach to Gaussian distributions and is, therefore, more generic in its application to different use cases. However, the consideration of running quantiles or the buffering of values is not part of this thesis and requires further research.

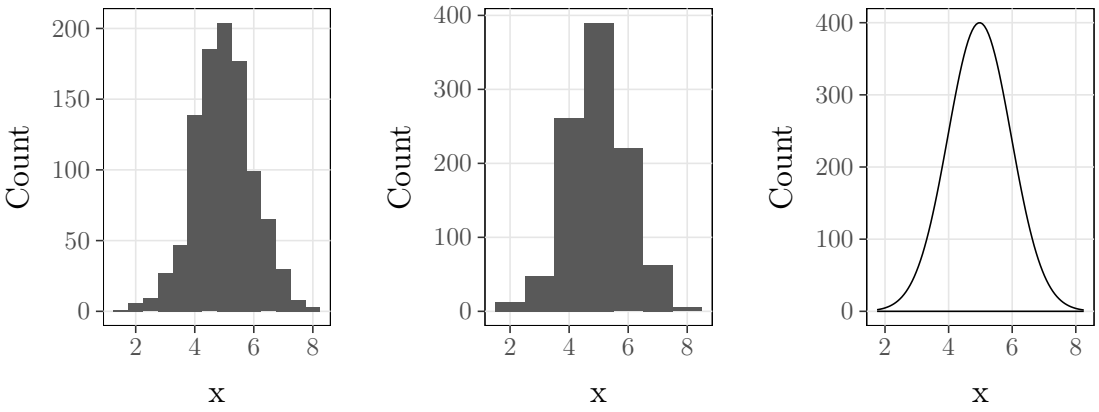
Discrete Multiset Statistics The second approach, the MultisetStatistics, targets a dynamic binning by transforming the given values. Depending on a given attribute, we consider a transformation function that maps a given value x to a specific bin. The statistics component only performs a counting for the given bins. This design enables a use case specific discretisation of values. For example, we can consider the transformation function of real values into integer values $f : \mathbb{R} \mapsto \mathbb{Z}$. This transformation function performs a binning of given attribute values. For each bin where an attribute value is assigned to, we only consider the count.

Although this approach is very intuitive it requires specific parameter-tuning. As the importance of different ranges of values for an attribute cannot be estimated automatically, transformation functions need to be specified for each attribute to ensure highest possible precision. The transformation function can be any arbitrary function, for example SAX (compare Section 5.1.2 on page 79), that defines the position and width of bins. Thus, this method is dependent on the actual use case and requires adequate initialisation for each attribute to be evaluated. However, as described in Section 5.3.1, the MultisetStatistics results in a loss of data due to transformation and only enables distance results in zero distance or maximum distance.

Characteristics of Approximation Figure 5.6 on the next page visualises the two proposed methods, incremental PDF statistics and MultisetStatistic, for the same input distribution. Figure 5.6a on the following page shows the initial distribution, from which values for both statistics are sampled. The initial distribution is a Gaussian distribution that is parameterised with a mean μ of 5 and a standard deviation σ of 1. From this distribution, 1000 values are picked randomly and fed to both statistics approaches.

The Figure 5.6b on the next page visualises the collected statistics for the Multiset-Statistics. The MultisetStatistics in this example utilises the transformation function $f : \mathbb{R} \mapsto \mathbb{Z}$. While the overall shape matches the input data, the limited granularity means that details of the initial distribution are lost. Figure 5.6c on the following page visualises the incremental PDF statistics. The underlying distribution is accurately replicated. While the MultisetStatistics stores statistics for 7 bins and therefore requires to store 14 values in total, the incremental PDF statistics only requires to store 3 values for one Gaussian distribution.

For further analysis of the proposed approaches, the generated statistics visualised in Figures 5.6b to 5.6c on the next page are used. This can be compared to the utilisation of a recorded identity profile (see Section 4.5.1 on page 67). However, in this use case we only consider attribute statistics. To analyse the characteristics, a second validation distribution is generated to check the resulting distance results. This validation distribution is generated with differing mean values. The values for sample size as well as standard deviation are fixed. For each iteration, the mean value is increased for 0.1 to influence the actual overlap



(a) Histogram of generated input distribution (b) Binning of MultisetStatistic (c) Distributions of incremental PDF statistics

Figure 5.6.: Approximated statistics for attribute values. Figure 5.6a visualises 1000 values generated randomly following a normal distribution with $\mu = 5$ and $\sigma = 1$ that are used as input. Figure 5.6b shows the binning of values via MultisetStatistic. For binning, the values are rounded to integer precision. A distribution-based approach, called incremental PDF statistics, is visualised in Figure 5.6c. It shows the approximated distributions.

of the initial distribution as well as the validation distribution. With decreasing overlap we can expect the resulting distance to grow. The optimal behaviour is, therefore, defined by

$$\text{dist}(a, b) = |a| - \text{overlap}(a, b) * |a|; |a| = |b| \tag{5.3}$$

where overlap defines a function to calculate the current overlap of the two given distributions a and b. To calculate the overlap we utilise the overlapping coefficient [113], that is the geometric overlap between the two distributions.

Next to the characteristics of overlap, we also analyse the behaviour for varying amounts of samples. We therefore generate a validation distribution with differing sample counts in the range from 0 to 2000. For each iteration the sample count is increased for 11. Values for mean as well as standard deviation are fixed. Therefore, we can expect the optimal behaviour to be defined as

$$\text{dist}(a, b) = \begin{cases} |a| - |b| & \text{if } x \leq |a| \\ -|a| + |b| & \text{if } x > |a|. \end{cases} \tag{5.4}$$

Figure 5.7 on the next page visualises the results for the analysis of differing overlap as well as sample count. The Figure on the left, Figure 5.7a on the facing page shows the characteristics for two distributions that share the same features. For one of the distributions, the mean is changed repeatedly to vary the overlap of the two distributions. Here, an overlap of 0 means, that both distributions have a relative distance of 1, whereas an overlap of 1 means, that we expect a distance of 0. For MultisetStatistics as well as incremental PDF statistics it can be seen, that they are close to the expected behaviour. Summarising, the parameterless incremental PDF statistics approach shows a slightly better precision than the MultisetStatistics.

In Figure 5.7b the influence on the number of randomly generated samples from a given distribution is visualised for the MultisetStatistics and incremental PDF statistics. When the randomly generated number of samples matches the number of samples from the given distribution, we expect a distance of 0. With increasing or decreasing number of samples we expect the distance to increase or decrease linearly. As can be seen in the visualisation, this expectation is approximated by both approaches. Still, the MultisetStatistics gives better results for the given use case. However, it can be seen that MultisetStatistics shows worst performance when the number of samples from validation distribution matches the original number of samples. This is caused by statistical variations of statistics within the single bins causing the resulting distance to grow. The incremental PDF statistics is more robust to the precise position of samples but produces smaller distances than expected for number of samples exceeding the original distribution.

Due to the incremental clustering of the incremental PDF statistics that approximates GMMs, the encoding is dependent on the order of events. Depending on the order of values, there is a chance that learned distributions differ for the same set of values. Following this, attribute distances based on approximated GMM can differ when comparing two trees T_1 and T_2, e.g. , $\text{dist}(T_1, T_2) \neq \text{dist}(T_2, T_1)$. Thus, the approximated GMM approach does not comply with the requirement of metrics to be symmetric.

Deviation of Approximation In the following, we approximate the deviation we need to expect from applying any of the proposed incremental approaches for SplittedStatistics to measure distances based on attribute values. Let $\mathscr{T} = (\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_n)$ be a statistical sample of dynamic trees. First, the distance between all possible pairs of dynamic trees are computed as

$$a_{j,k}(\mathscr{T}) = \text{dist}^{\text{attrib}}(\mathscr{T}_j, \mathscr{T}_k), j, k = 1, 2, \ldots, n, \tag{5.5}$$

where $\text{dist}^{\text{attrib}}$ denotes the distance based on SplittedStatistics that provides a distance for two dynamic trees. To not bias the actual analysis, the distance only considers attribute events for distance calculation. Thus, it ignores all vertex events for the given trees.

Given that distance matrix A, we calculate the *mean relative deviation* $\bar{e}$ for the given distance $\text{dist}^{\text{attrib}}$ regarding two characteristics of metrics: the identity and symmetry. For distance metrics it is required, that the distance of an object to itself is 0, that is the identity of indiscernibles. Thus, we can expect the distance of a dynamic tree $\mathscr{T}_1$ to itself to be 0, that is $\text{dist}^{\text{attrib}}(\mathscr{T}_1, \mathscr{T}_1) = 0$. Therefore, to analyse deviation regarding this characteristics of a metric, we consider all distances on the diagonal of the matrix as the actual deviation. Thus, we calculate the *mean relative identity deviation* regarding the identity of our proposed distance measure by

$$\bar{e}^{\text{diag}} = \frac{1}{n} \sum_{i=1}^{n} \frac{a_{i,i}}{2\langle \mathscr{T}_i | \mathscr{T}_i \rangle}. \tag{5.6}$$

Another characteristic of a distance metric is the symmetry. Hence, the distance between two dynamic trees $\mathscr{T}_1$ and $\mathscr{T}_2$ is independent from the order of the two trees for a given distance measure, that is $\text{dist}^{\text{attrib}}(\mathscr{T}_1, \mathscr{T}_2) = \text{dist}^{\text{attrib}}(\mathscr{T}_2, \mathscr{T}_1)$. As the exact distance for the two dynamic trees is not known, we consider the mean distance of the two trees to be

the expected distance value, that is

$$\overline{\mathrm{dist}(\mathscr{T}_1, \mathscr{T}_2)} = \frac{\mathrm{dist}(\mathscr{T}_1, \mathscr{T}_2) + \mathrm{dist}(\mathscr{T}_2, \mathscr{T}_1)}{2}. \tag{5.7}$$

Given the expected distance in Equation (5.7) we can calculate the *mean relative symmetry deviation* regarding the symmetry of our proposed distance measure by

$$\bar{e}^{\mathrm{symm}} = \frac{1}{n^2 - n} \sum_{j=2}^{n} \sum_{k=1}^{j-1} \frac{|a_{j,k} - a_{k,j}|}{\langle \mathscr{T}_j | \mathscr{T}_j \rangle + \langle \mathscr{T}_k | \mathscr{T}_k \rangle}. \tag{5.8}$$

Based on a statistical sample of $n = 1000$ batch jobs with sizes not smaller than 1000 vertices, we examined a mean relative symmetry deviation $\bar{e}^{\mathrm{symm}}$ of $0.001 \pm 3.541 \times 10^{-5}$ and a mean relative identity deviation $\bar{e}^{\mathrm{diag}}$ of 0.005 ± 0.001 for incremental PDF statistics.

Figure 5.8 on the facing page visualises the influence of varying weighting of attribute distance to vertex distance and varying tree sizes for incremental PDF statistics. With increased weight of attribute distances in the overall distance calculation the deviation increases linearly (compare Figure 5.8a). Attribute values are sparsely distributed within the trees. With growing number of shared identities we expect attribute values to become more relevant statistically and thus, we expect the representation of distribution by SplittedStatistics to improve. This assumption is supported by Figure 5.8b: With increased tree sizes the relative deviation decreases.

The same analysis for MultisetStatistics can be found in the Appendix in Figure C.1 on page 177. As the MultisetStatistics does not depend on the order of events to learn underlying statistics, deviations are negligible. In fact, any deviation that might appear for MultisetStatistics is caused by rounding errors.

5.3.2. Incremental Dynamic Distances

The two distribution-based approaches MultisetStatistics and incremental PDF statistics provide representations of attributes for dynamic trees. To approximate tree distances based on the proposed methods, we use projections of attribute distributions for two given objects. The distributions are generated by appropriate identity profile projection operators that depend on a given SplittedStatistics, either MultisetStatistics or incremental PDF statistics, as well as the specific statistic under consideration. For example, the multiplicity identity profile projection operator M is defined on MultisetStatistics and the statistic *count* of an object. In fact, the SplittedStatistics are a generalisation of a simple counting approach (see Section 5.2.3 on page 83).

The process to create attribute distributions supplementing identities is a two-step procedure. First, the identity is determined as a prerequisite to identify collected statistics of related distributions. Given this, the overlap of attribute statistics can be determined based on the updated statistics of the observed object. This update and overlap determination is also defined incrementally: When an existing distribution of the observed identity profile is extended, each new value is checked to still fit the recorded identity profile as stated by identity profile projection. This determines the distance for the event under consideration. A high-level overview of the functioning is visualised in Algorithm 4. Please note that the distance result depends on the underlying statistics component. For MultisetStatistics only minimum or maximum distance are returned while incremental PDF statistics allows for continuous distance values.

Algorithm 4 Attribute-based distance based on SplittedStatistics

Precondition:
`x` is any value that is added to the observed tree $\mathscr{T}'$
Id is the associated identity of the composite vertex
$\mathscr{T}$ is the recorded tree
Postcondition: Attribute-based `distance`

```
1: function UpdateAttribute(Id, x)
2:     statistic′ ← get statistic of Id for 𝒯′
3:     Add(x) to statistic′                          ▷ See Algorithm 2 on page 91
4:     statistic ← get statistic of Id for 𝒯
5:     if BinValue(statistics′, x) ≥ BinValue(statistics, x) then
6:         distance ← mismatch
7:     else
8:         distance ← match
9:     return distance
```

Example Distances

In dynamic trees, various statistics can be considered important to represent the dynamics of the tree and attributes. We already consider simple statistics such as the multiplicity of identities M. We extend the available identity profile projection operators to support the distribution of attribute values by introducing the identity profile projection operator Λ. Similar to the identity profile projection operator M, it represents the distribution of attribute values by means of multiplicity. However, it allows to distinguish between the semantics of attributes and vertices.

The utilisation of the identity profile projection operators 1, M, and Λ do not represent dynamics of dynamic trees. We therefore introduce further non-trivial statistics. Non-trivial statistics focus on the tree dynamics to enable a more flexible representation and thus potentially a better separability. In the following, we briefly discuss two non-trivial statistics to derive attribute distributions. We focus on the lifetime of a vertex by considering its start and exit event as well as the frequency.

Example 5.8. Trees that are equal in structure can still differ in their dynamic behaviour. Consider a local process spawned to query a remote service for the location of a file in a distributed storage. The natural latency of the request likely makes both traffic and transfer rate insignificant. However, if the remote service is not available with regular performance, the local process will take significantly longer to succeed with its request. In contrast to traffic and transfer rates, lifetime must be calculated from start and end of a vertex.

The lifetime of a vertex encodes its duration as the delay between the start event of a specific vertex v, $\varnothing \to v$ and the exit event, $v \to \varnothing$. This statistic expresses the relation of the underlying identity on a local scale: lifetime is derived for each vertex separately, even if concurrent vertices share the same identity. The lifetime d itself does not necessarily depend on the lifetime of the children of v, d^{children}, that is $d \geq d^{\text{children}}$. The consideration of a lifetime of an object represents a realistic use case:

Example 5.9. Consider the utilisation of a bash script to execute tasks within a job in a batch system. The bash script can describe entirely different tasks such as the preparation

of the working environment or the execution of a complex data analysis task. Both scripts result in two processes named equally. Yet, the duration of both tasks is distinctly different. Thus, we need to differentiate between both tasks, even though they share the same identity.

Therefore, we introduce the duration identity profile projection operator D as a distribution that is defined on SplittedStatistics utilising the statistic of duration. Thus, the identity profile projection operator D is given by $\langle \mathscr{T}_1 | D | \mathscr{T}_2 \rangle$ and ensures the overlap of two given identity profiles based on the distribution of durations of distinct identities.

Another important statistic to consider in the domain of dynamic objects is the presence of repeating patterns. Repetition can be defined as the delay between a given type of events for a specific identity. We therefore introduce a frequency identity profile projection operator F as a distribution defined on SplittedStatistics utilising the frequency of identities, that is $\langle \mathscr{T}_1 | F | \mathscr{T}_2 \rangle$. The frequency identity profile projection operator expresses the relation of identities on a global scale: frequency is derived from two vertices of the same identity, disregarding any intermediate siblings not reflected in the identities. Whenever a recurrent pattern can be found in data, it has the characteristic to be invariant to shifts [155] and therefore ensures a robust feature data analysis.

5.4. Summary

In this Chapter we focused on incremental tree distances taking attributes into account. Thus, our evaluation of existing distances put special attention on approximation methods to handle attributes in tree-structured data. Our evaluation revealed that most existing approaches either focus on a semantic interpretation of textual attributes, or the modelling of attributes and values as common tree vertices. Such approaches eliminate any need for special handling of attributes.

However, attributes in dynamic, streaming trees require special consideration as not only the mapping of one attribute value per vertex needs to be ensured, but a time series of values. Based on our analysis of advantages and disadvantages of existing models, we introduce a generic representation of attributes for dynamic, streaming trees.

The proposed generic representation of attributes for dynamic, streaming trees integrates seamlessly with the existing framework for tree distance measures introduced in Chapter 4 on page 39. Our proposed approach is distinguished from existing approaches by introducing two differing types of vertices to model dynamic trees with attributes. In contrast to the uniform modelling of nodes, attributes and values, our specific modelling enables a generalisation of different types of attributes and values. Furthermore, this approach allows to express complex relationships based on attributes and their dependency on specific vertices.

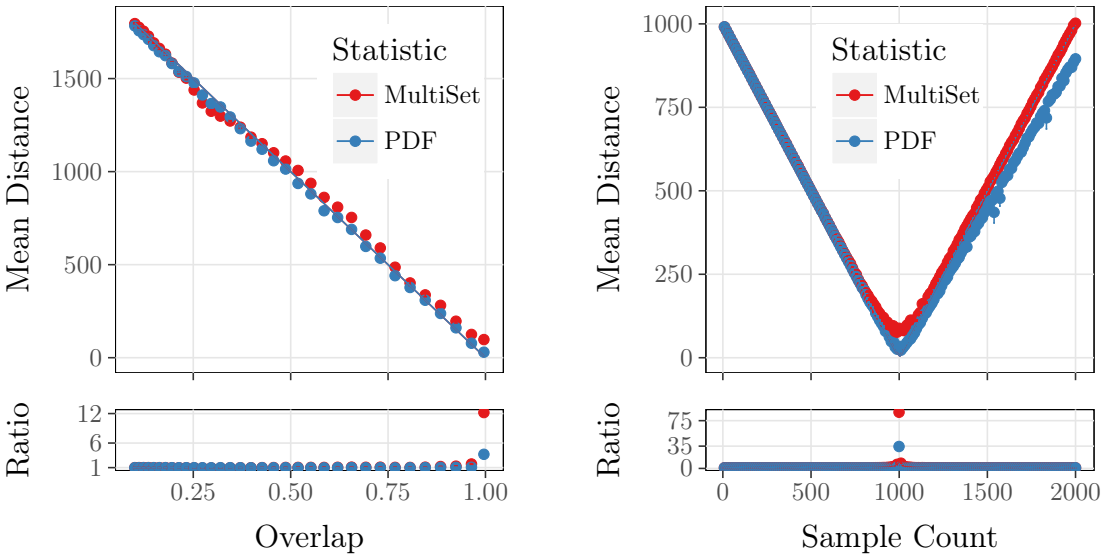
In specific we introduce the concept of composite vertices as well as atomic vertices. While composite vertices model regular objects in the tree and ensure the integration of the underlying framework, atomic vertices model the attributes of an object represented by a specific composite vertex. The atomic vertices carry the non-trivial attribute values modelled as an associated vertex weight. Thus, the modelling of attributes does not formally define values as vertices, but allows for a separate handling.

To represent the values of attributes and enable distance measurement, we focus on statistical decomposition into proper distributions of values. As the underlying framework targets compression of identities, we explicitly enable the representation of groups of values via distributions. This distribution-based representation is not limited to the plain values

of an attribute. Instead, we distinguish local and global attributes supporting categorial, discrete, and continuous values: Local attributes mainly focus on the representation of attribute values of individual and overlapping identities. In addition, global attributes enable the derivation of more complex measures such as statistics on multiplicity, frequency, or lifetime of vertices.

In specific, we propose two methods to represent underlying attribute distributions: MultisetStatistics and incremental PDF statistics. While MultisetStatistics ensures properties of a pseudo-metric, incremental PDF statistics focuses on optimisation of space complexity and generic applicability for different data types and ranges. However, the two are a tradeoff between robustness and accuracy: incremental PDF statistics provide smooth, accurate representations if the type of distribution, such as a Gaussian, is known. MultisetStatistics require a fixed resolution, but easily represent arbitrary shapes, including skewed data.

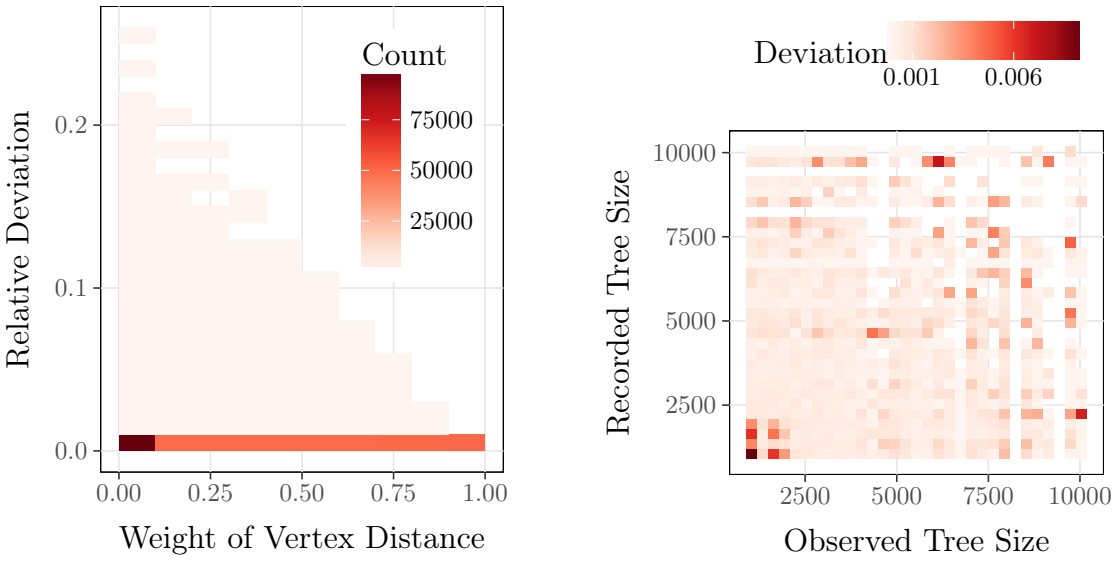
In summary, our approach to representing attributes integrates smoothly into our existing formalisation, thus implicitly enabling distance-based projection. The distance is expressed by considering the similarity or difference based on the overlap of distributions. Contrary to approaches from literature, this approach allows continuous distance measurements for any attributes with an underlying order, most importantly numerical attributes.



(a) Attribute-based distance for varying overlap

(b) Attribute-based distance for varying number of samples

Figure 5.7.: Visualisation of characteristics of incremental PDF statistics and MultisetStatistics for distributions with varying overlap and sample count. Figure 5.7a visualises the distances for a validation distribution with overlap in the range from 0 % to 100 %. For an overlap of 100 % we expect a distance of 0, while we expect a maximal distance for an overlap of 0 %. It can be seen, that incremental PDF statistics shows slightly better results. Figure 5.7b shows the characteristics of both statistics approaches by varying the number of samples of the validation distribution. The count is varied from 0 % to 200 %. This analysis shows that MultisetStatistics overall shows a better performance except for the same amount of values initially learned (compare Figure 5.6b on the facing page). For 1000 values the distribution is statistically not stable to match the exact counts of given bins, thus we get higher distances than expected. The incremental PDF statistics shows good performance for values up to 100 % and results in lower distances than expected for higher counts of samples.



(a) Relative deviation e of distance result with regard to attribute influence

(b) Mean relative deviation $\bar{e}$ of distance result for different tree sizes due to attributes

Figure 5.8.: Influence of attribute weighting and tree size to the relative deviation e for incremental PDF statistics of calculated distance result for dynamic trees. Statistics have been calculated from 10 independent, randomly selected samples of 100 dynamic trees. The original dataset has been filtered to include only dynamic trees with sizes of $1000 < max(|\mathscr{T}_i|), \forall \mathscr{T}_i \in \mathscr{T} < 10\,000$. Figure 5.8a shows influence of the weighting factor on the relative deviation e of the distance result. It can be seen, that weight and relative deviation are linearly correlated. When the influence of vertex identities is 0, only attributes affect the distance. Thus, the relative deviation is at its maximum, when the attributes are weighted highest. Figure 5.8b shows the influence of the tree size on the mean relative deviation $\bar{e}$ of the distance results for a vertex weight of 0. As statistics in the underlying statistics component get more significant with more available samples, the mean relative deviation decreases.

6. Superposition of Dynamic Tree Features

There is a multitude of approximating tree distance approaches, each with a representation of trees resulting in a different interpretation of differences and similarities (see Section 4.1 on page 39). Our own approach is capable of covering a variety of features, some of which cannot be represented by common approaches. However, there is no single correct tree distance, as relevant features of trees, vertices, and attributes strongly depend on the use case.

In fact, black-box use cases such as ours, the online analysis of batch jobs in production batch systems, do not provide prior information on features that are relevant in the first place. We must rely on explorative data analyses and models based on a high-level understanding of the given use case. Most critically, this means that we might overestimate the relevance of features and robustness of their description.

For example, previous analysis of finite-length and infinite-length methods for tree decomposition show complementary advantages and disadvantages (see Section 4.4.5 on page 66). While infinite-length encoding allows a fast identification of *equal* ancestry, they cannot express *similar* ancestry. Consequently distances to similar trees may be overestimated. Identifying partially equal ancestry is better covered by finite-length encoding approaches, which focus on identification of patterns within a tree. Therefore, hierarchical dependencies become less important as the localisation of patterns in the tree is usually not covered. This results in distance perturbation with a tendency to overestimate similarity of trees (see Section 4.4.4 on page 58).

Without prior knowledge, it is impossible to tell which measure is feasible. Additionally, we have no guarantee that any individual measure is appropriate by itself. This is especially the case with our data being non-stationary, meaning that differing features may become important over time.

Thus, we desire a method to assess multiple features in parallel. Notably, the goal is not to merge multiple measures to a single one, as this implies dependencies that we cannot estimate. Instead, multiple simple measures should be loosely coupled to be robust when individual assumptions do not hold. In the following we propose to combine multiple identity classes to an identity ensemble class to enable flexibility while retaining precision and accuracy of individual methods.

6.1. Related Work

Ensemble methods are often used for classification. Classification is a method of supervised learning with the goal to identify distinct classes in a given dataset. In this context, classifiers are trained to categorise data as belonging to specific classes based on a set of distinct features. Ensembles combine a set of classifiers in order to classify data with regard to multiple features. Typically, classifiers are combined by weighted or unweighted voting. The construction of expressive ensembles of classifiers is an active research area.

This is mainly justified by the discovery that ensembles are often more accurate and robust than individual classifiers [76, 111]. In practice, a number of classifiers are built to combine the results of the different classifiers into a common decision. One key concept of an ensemble is the requirement of diversity [124]. That is, each classifier should cover distinct features.

Diversity in an ensemble can be achieved by: selecting different algorithms to build a classifier [144], changing the training data of the base classifier by sampling or directed replication [47], selecting different parameters to train each classifier, or modifying the classifiers internally either by re-weighting the training data or by randomisation [191].

Recent methods of ensemble schemes consider online approaches that adapt to non-stationary data by iteratively evaluating accuracy of predictions [40] or constant deletion, creation, or modification of classifiers based on their performance [111].

While we must also deal with non-stationary data, the online adaptation of ensembles is undesirable in our use case: the distance between trees must be guaranteed to be well-defined over time. Otherwise unexpected behaviour can be expected from subsequent analyses. We therefore restrict our approach to a lightweight ensemble-based classifier building on preferably orthogonal properties.

6.2. Encoding Identity Ensembles

To realise robust distance measurement for trees especially in situations of non-stationary data and noise, we introduce an ensemble-based identity profile projection strategy based on our framework. That is, a number of independent identity classes are used to describe each entity of a tree and subsequently the tree as a whole.

Unlike attributes, which are a refinement of identities, ensembles are orthogonal, using separate identity classes independently. In effect, this means that several independent identities are assigned to each vertex. However, this does not translate trivially to the representation of identity profiles, which may aggregate individual identities differently.

6.2.1. Ensemble Encoding

Ensembles relate each vertex to multiple identities at once. An overview about the concept of identity ensembles is given in Figure 6.1 on the next page. There are no additional dependencies, allowing pre-calculated identities to be reused. In addition, ensembles work orthogonal to identities: vertices can be encoded separately for each identity class, without interference. This naturally preserves complexity with regards to the number of vertices.

Definition 6.1 (Identity ensemble)**.** The *identity ensemble* Id^{e} of a vertex $v \in \mathrm{V}(T)$ is a collection of identities of the given vertex v.

An identity ensemble $\mathrm{Id}^{\mathrm{e}}(v)$ is defined as

$$\mathrm{Id}(v)^{\mathrm{e}} = (\{|\mathcal{P}(v), \mathcal{Q}(v), \mathcal{S}(v), \mathcal{V}(v)\rangle_1, \ldots, |\mathcal{P}(v), \mathcal{Q}(v), \mathcal{S}(v), \mathcal{V}(v)\rangle_m\}) \tag{6.1}$$

where m is the number identities $|\mathcal{P}, \mathcal{Q}, \mathcal{S}, \mathcal{V}\rangle$ to be used for the identity ensemble. Following this definition, the identity ensemble of a tree T is, therefore, described as

$$\mathrm{Id}^{\mathrm{e}}(T) = (\{\mathrm{Id}^{\mathrm{e}}(e) \,|\, e \in S(T)\}). \tag{6.2}$$

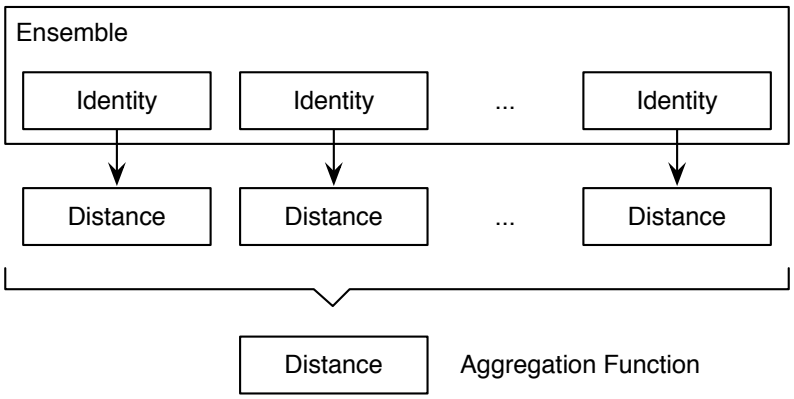


Figure 6.1.: Schematic visualisation of the concept of an identity ensemble. An identity ensemble is composed of a number of identity classes. For each identity class an identity is created. For each identity the appropriate identity projection is determined. Thus, several independent distances are aggregated to one final distance that is considered for further evalutation.

Note that similar to identity profiles, the identity ensemble is intentionally defined as an arbitrary collection of identities. It is left for further specification and implementation to decide on the underlying type of collection, either ordered or unordered collection.

When defining an identity ensemble special care should be taken to select identities with orthogonal characteristics. An appropriate choice of identities allows to reflect multiple features at once. This can be complementary features, such as local and global similarities with finite and infinite-length encodings, which are robust to changing characteristics of data in non-stationary environments. Furthermore we expect results to become more accurate when compared to a single identity.

Still, ensembles can be detrimental if chosen inappropriately. Most importantly, additional identity classes have an impact on processing and memory, even though the complexity in regards to vertices stays the same. Furthermore, using identity classes unsuitable for a use case deteriorates the overall results.

6.2.2. Ensemble-Based projection

While identity ensembles are straightforward at vertex level, combining them to tree identity profiles is not unambiguous. Most prominently, different identity classes do not necessarily yield the same number of identities for a given tree. As such, there are several options to aggregate identities to form identity profiles, and subsequently, to incrementally handle the identities of individual vertices. In the following, we discuss the available options and refer to advantages as well as disadvantages.

Independent Encoding

The most straightforward method of handling identities is an *independent encoding of identity ensembles*. This encoding considers not just identities as independent, but identity profiles as well. The identity ensemble profile is, therefore, given by

$$|\widetilde{T}\rangle = (\{|T\rangle_1, \ldots, |T\rangle_m\}). \tag{6.3}$$

Thus, the space complexity of the identity ensemble profile is $\mathcal{O}(m \cdot n^{\bar{f}}), \bar{f} \leq 1$ (see Section 4.5.3 on page 73) for m different identity classes within the specified identity ensemble. This preserves the expected sublinear space complexity, and also converges towards constant complexity for trees with high fanout.

Each of the generated identities is evaluated with regard to the respective streaming and recorded identity profile. Therefore, the approach directly follows the established identity profile projections considered so far. The independent handling of identities results in a collection of parallel distance results, each treated independently from the others.

Independent encoding of identity ensembles does not consider correlation of information of identities of the same vertex. For example, assume the two identity classes Id^{P} and $\mathrm{Id}^{\mathrm{pq}}$, that is infinite-length and finite-length encoding. This identity class Id^{P} maps the entire ancestry of vertices, but not siblings. In contrast, the identity class $\mathrm{Id}^{\mathrm{pq}}$ only considers local ancestry, but for both parents and siblings. If for a vertex $v \in \mathrm{V}(T)$ a sibling is changed compared to a tree T', the mapping within the tree T for each identity can occur at completely different positions than in T'. While the identity of Id^{P} considering $\mathcal{P}$ dimension ensures correct location, the finite-length identity $\mathrm{Id}^{\mathrm{pq}}$ can be arbitrarily repositioned to a similar context.

Dependent Encoding

To limit the influence of independently positioned identities within one identity ensemble, one can strictly combine different identity classes. We refer to this option as *dependent encoding of identity ensembles*. Dependent encoding of identity ensembles is based on an identity profile projection regarding the set of *all* given identities within the identity ensemble, that is

$$|\widetilde{T}\rangle = \sum_{e \in S(T)} (\{|\mathcal{P}(e), \mathcal{Q}(e), \mathcal{S}(e), \mathcal{V}(e)\rangle_1, \ldots, |\mathcal{P}(e), \mathcal{Q}(e), \mathcal{S}(e), \mathcal{V}(e)\rangle_m\}), \tag{6.4}$$

where m denotes the number of identity classes of the given identity ensemble. With this encoding, an identity ensemble only matches when all individual identities match. Therefore, very specific rules can be deployed to evaluate distances for tree-structured data.

However, the dependency is a severe constraint on identity ensembles. The encoding is not flexible in measuring varying relevant features, but requires all features to be relevant. Furthermore, the dependency of identities limits compression of similar identities. A dependent identity ensembles can only compress two vertices if all identities match. The chance for equal combinations of identities decrease exponentially with the number of identity classes considered. Thus, the space complexity is worse compared to independent encoding of identity ensembles.

Nested Encoding

To improve space complexity of dependent encoding, one can constraint dependencies to a *nested encoding of identity ensembles*. Nested encoding creates an ordered dependency between identity classes. Regarding the example with identity classes Id^{P} and $\mathrm{Id}^{\mathrm{pq}}$, this means we use the identity of the position-specifying identity Id^{P} to preselect matches for the identity $\mathrm{Id}^{\mathrm{pq}}$. In other words, we restrict the possibilities for mapping remaining identities of a given identity ensemble. Consequently, we have the possibility to consecutively narrow the locality or context of generated identities. An overview of the nested encoding strategy is

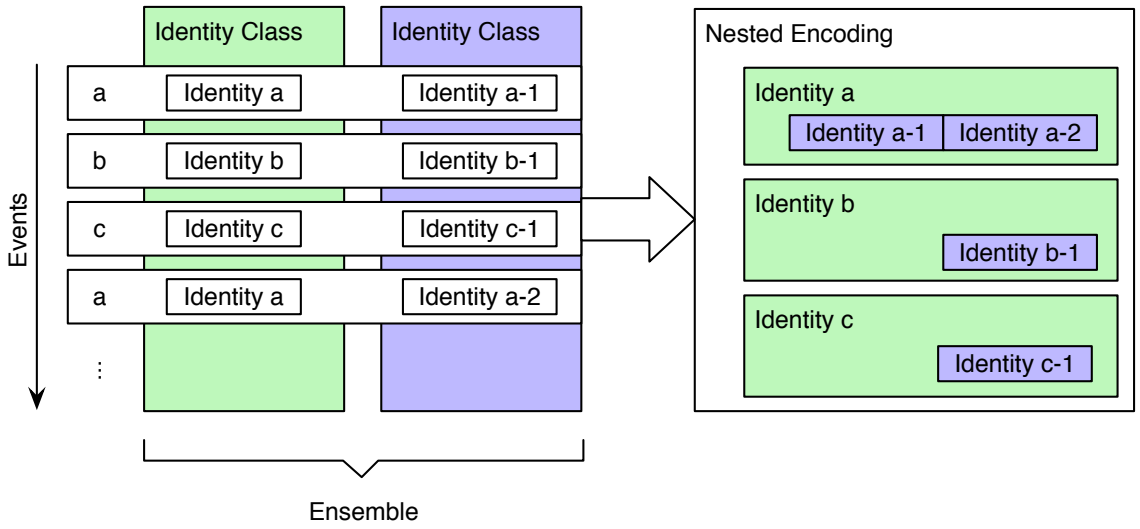


Figure 6.2.: Overview of nested ensemble encoding. The example defines an identity ensemble from two differing identity classes. Green boxes denote the *outer* identity class, whereas blue boxes denote the *inner* identity class. For each event ("a", "b", "c", "a", ...) respective identities are generated. With regard to equality of identities of outer identity class, the identities of the inner identity class are grouped. For example the identity created for the event entitled "a" creates a collision for the outer identity class. However, the inner identity class differentiates between the two events. Consequently, the inner identities "a-1" and "a-2" are grouped into the identity "a".

visualised in Figure 6.2. For two identity classes that build an identity ensemble, respective identities are generated. The order specified in the identity ensemble defines the nesting. Therefore, identities of the inner identity class are mapped with regard to outer identities. This results in a grouping of inner identities for equal identities on outer level.

However, nested encoding of identity ensembles comes with two disadvantages: Although space complexity is better than dependent encoding of identity ensembles, it is still worse than the independent encoding strategy. As the nesting is directly encoded, we do not benefit much from compression. While outer identities can have collisions with any vertex of the tree, subsequent identities are restricted to the parent identity vertices. Only if outer identities are more generic than inner identities we can expect compression.

The second disadvantage is about the hierarchy of nesting. The first level identity specifies the lookup key for matching further identities from remaining identity classes of the given identity ensemble. Thus, a too specific first level identity, for example Id^{P} might not be existent in a given identity ensemble profile of a tree. Whenever the identity for Id^{P} does not exist in a given identity ensemble profile no further mapping of remaining identity classes within the identity ensemble can be evaluated. Thus, nested encoding of identity ensembles is sensitive to the order of given identity classes. For example, it can be more efficient to utilise the identity $\mathrm{Id}^{\mathrm{pq}}$ to access further identities that can in the following be used to raise accuracy of distance evaluation. Consequently, the mapping of nested encoding is ambiguous and requires special care to select the order of identity classes.

Nested Independent Encoding

Further options such as nested *and* independent encoding of identity ensembles can be considered. As identity ensembles can work equivalently to identity classes, this allows having identity ensembles of identity ensembles. However, such options are vastly too specific for our use case.

6.2.3. Summary

In summary, there are different options to encode identity ensembles, each targeting different use cases. Independent encodings allow testing different identity classes in parallel, but are not capable of expressing correlations between them. In contrast, dependent encodings allow combining multiple identity classes, but cannot express partial similarities.

For our use case, we have only limited knowledge of the underlying rules of trees. Most importantly, the models we use are based on estimates of isolated features. As such, we cannot assume any specific correlation between models. This matches the behaviour of independent encoding identity ensembles.

Additionally, an independent encoding of identity ensembles preserves the space complexity of the underlying identity classes. We consider this most crucial in streaming environments. As such, we restrict further evaluations to the application of independent encoding identity ensembles.

6.3. Ensemble-Based Tree Distances

When utilising identity ensembles, we exploit the incremental behaviour of distance measures for dynamic trees in streaming environments (see Section 4.5.4 on page 73). Furthermore, an independent encoding of identity ensembles also makes the distance calculation for each identity class independent. Consequently, for each identity in an identity ensembles we can incrementally calculate the similarity to another tree or vertex, without regard to parallel identities.

Thus, the identity ensemble profile projection is composed of the results from the identity profile projection for each identity class. However, using an independent encoding for the identity ensemble does not define how to merge the results to a single distance or similarity value. Consequently, we introduce an *aggregation function* ϕ to combine the different similarity measures of identity profile projections. Thus, the absolute distance for two trees $\mathscr{T}_1$, $\mathscr{T}_2$ (see Equation (4.24)) is given by

$$\begin{aligned}\operatorname{dist}(\mathscr{T}_1, \mathscr{T}_2) = \langle\widetilde{\mathscr{T}_1}|\theta|\widetilde{\mathscr{T}_1}\rangle + \langle\widetilde{\mathscr{T}_2}|\theta|\widetilde{\mathscr{T}_2}\rangle - 2 \sum_{i=1}^{|S(\mathscr{T}_2)|} \phi(\langle\widetilde{\mathscr{T}_1}|\theta|\, \mathcal{P}_{2,i}, \mathcal{Q}_{2,i}, \mathcal{S}_{2,i}, \mathcal{V}_{2,i}\rangle_1, \ldots, \\ \langle\widetilde{\mathscr{T}_1}|\theta|\, \mathcal{P}_{2,i}, \mathcal{Q}_{2,i}, \mathcal{S}_{2,i}, \mathcal{V}_{2,i}\rangle_m), \quad (6.5)\end{aligned}$$

where the aggregation function ϕ can be any arbitrary function to combine the different results from identity profile projection of each identity class.

Given the absolute distance for the identity ensemble profile projection, we can accordingly derive a relative distance (see Equation (4.25)) as well as an incremental distance (see Equation (7.6)). The relative distance is given by

$$\operatorname{dist}(\mathscr{T}_1, \mathscr{T}_2) = 1 - \frac{\phi(\langle\widetilde{\mathscr{T}_1}|\theta|\mathscr{T}_2\rangle_1, \ldots, \langle\widetilde{\mathscr{T}_1}|\theta|\mathscr{T}_2\rangle_m)}{\langle\widetilde{\mathscr{T}_1}|\theta|\widetilde{\mathscr{T}_1}\rangle + \langle\widetilde{\mathscr{T}_2}|\theta|\widetilde{\mathscr{T}_2}\rangle - \phi(\langle\widetilde{\mathscr{T}_1}|\theta|\mathscr{T}_2\rangle_1, \ldots, \langle\widetilde{\mathscr{T}_1}|\theta|\mathscr{T}_2\rangle_m)}, \quad (6.6)$$

and finally the incremental distance is given by the recursion formula

$$\text{dist}_0 = \sum_j \alpha_j \langle \widetilde{\mathscr{T}_1} | \theta_j | \widetilde{\mathscr{T}_1} \rangle, \text{ with } \sum_j \alpha_j = 1$$

$$\text{dist}_i = \text{dist}_{i-1} - \phi \left(\biguplus_m \sum_j \alpha_j (2 \langle \widetilde{\mathscr{T}_1} | \theta_j | \, \mathcal{P}_{2,i}, \mathcal{Q}_{2,i}, \mathcal{S}_{2,i}, \mathcal{V}_{2,i} \rangle_m - 1) \right). \tag{6.7}$$

6.3.1. Distance Aggregation Function

The choice of an aggregation function dictates how the results of independent identity profile projections of identities are merged. Notably, this is distinct from the notion of encoding identities together in an identity ensemble. Even if the result of each identity profile projection is strictly linked, for example by multiplication, an independent encoding of identities still allows for separate compression of each identity profile.

For simplicity, we desire an aggregation function to have the same domain and codomain as the identity profile projection of identities. As such, the combination of similarity measures can be expressed by an aggregation function defined as follows [35]:

Definition 6.2 (Aggregation function)**.** An *aggregation function* ϕ is a function of $m \geq 1$ arguments that maps the value range of given arguments onto the unit interval $\phi : [0,1]^m \to [0,1]$.

Notably, this definition also preserves other features of the initial identity profile projections. Most importantly, any function that is monotonous with respect to any identity profile projection is also monotonous with respect to an aggregation function of any identity profile projection.

For our use case, we desire an aggregation function that satisfies the following conditions: First, it must be robust to any individual identity class being too specific to match across trees. Second, it must be robust to any individual identity class being too generic to ever not match across trees. Finally, it should not suppress any valid identity classes.

Without loss of generality we assume the aggregation function ϕ to be the arithmetic mean of the given similarity measures from specified identity ensemble profile projection, that is

$$\bar{X} = \frac{1}{n} \sum_{x \in X} x, \tag{6.8}$$

where X is a collection of n real values $x_i \in [0,1]$, $X = (\{x_1, \ldots, x_n\})$.

6.3.2. Auxiliary Identities

While we do have a model of the monitored jobs, it is based on high level specifications. As such, any expectations on hierarchy and attributes are only estimates. On the one hand, we have to assume that implementations deviate from our model. On the other hand, the execution environment can lead to nondeterministic differences.

However, our identity-based approach is fundamentally designed for precise identification. Thus, even small deviations can have exaggerated impact on calculated distances. Yet, always ignoring such deviations can severely underestimate distances. As such, in addition to identities conforming to a precise model, auxiliary identities that use a less restrictive model are desirable.

Towards Use Case Independence

The previously introduced identity class Id^{P} (see Section 4.4.4 on page 63) can exaggerate distances from the renaming of vertices. Especially when tackling non-stationary data, identity ensembles can have several advantages over single identities: they are easy to scale and parallelise, they can adapt to changes by leveraging different features of trees, they can quickly be adapted by pruning under-performing parts of the ensemble, and they therefore usually generate more accurate results. In the following we discuss the possibility to approximate the expected outcome by considering an ensemble-based approach to measure the distance for two trees T_1 and T_2.

Example 6.3. Consider a simple tree event stream representation $S^{\mathrm{simple}}(T)$ with $\sigma(\mathrm{V}(T))$ in preorder. Given two trees T_1 and T_2, with their symmetric difference $\mathrm{V}(T_1) \triangle \mathrm{V}(T_2) = \{T_1.root, T_2.root\}$, that is both trees are equal except for their root vertices. An identity profile projection of the identities derived from the definition $\mathrm{Id}^{\mathrm{P}}(e) = |\mathcal{P}(e), \mathcal{Q}(e) = \varnothing, \mathcal{S}(e) = \varnothing, \mathcal{V}(e)\rangle$, on both trees T_1 and T_2, we obtain the maximum distance regarding the two identity profiles $\mathrm{Id}^{\mathrm{P}}(T_1)$ and $\mathrm{Id}^{\mathrm{P}}(T_2)$. This is due to the fact, that the root vertex is recursively encoded into each identity. Therefore, the overlap of both identity profiles is 0, that is $\mathrm{Id}^{\mathrm{P}}(T_1) \cap \mathrm{Id}^{\mathrm{P}}(T_2) = \varnothing$. Consequently, the distance between both trees T_1, T_2 is maximal.

Receiving a maximum distance for two identity profiles $\mathrm{Id}^{\mathrm{P}}(T_1)$ and $\mathrm{Id}^{\mathrm{P}}(T_2)$ when only the root vertex of both trees is changed is consistent with identifying vertices by their ancestry. However, for TED we would only expect a distance of 1 in total.

Our approach to distance measurement for trees considers the ancestry of a vertex as a central characteristic to define its identity, compare Section 4.4.3 on page 54. However, it seems also reasonable to assume that the influence of a parent vertex onto its descendants decreases with increasing path lengths.

To accommodate this, one can use an identity ensemble class that builds on both, Id^{P} and $\mathrm{Id}^{\mathrm{pq}}$:

$$\mathrm{Id}^{\mathrm{e}}(T) = (\{\mathrm{Id}^{\mathrm{P}}(e), \mathrm{Id}^{\mathrm{pq}}(e)\})_{e \in S(T)}. \tag{6.9}$$

This identity ensemble class combines the features of both identity classes Id^{P} as well as $\mathrm{Id}^{\mathrm{pq}}$. The disadvantage of dynamic pq-gram identity profiles, for example the distortion by diamonds, are compensated by Id^{P}. In addition, the disadvantage of identity class Id^{P} to only identify exact matches for a given ancestry is compensated by flexibility of dynamic pq-gram identity profiles. Thus, we expect the distances to more accurately describe expectations by combining both identity profile projections. However, as mentioned before, memory requirements increase by combining two identity profile projection schemes.

Figure 6.3 on the next page visualises behaviour of Id^{e} as well as its components. The Id^{P} identity is only capable of expressing a limited range of differences. After 20 % of different vertices, the identity cannot distinguish changes anymore. In contrast, $\mathrm{Id}^{\mathrm{pq}}$ is not capable of distinguishing the depth of changes, as signified by its smaller error band. However, Id^{e} exhibits features of both identities: It is expressive over the entire range of differences, but also reflects how changes at different depths imply different impact.

The independent encoding of identity ensembles combining Id^{P} and $\mathrm{Id}^{\mathrm{pq}}$ robustly measures changes in the ancestry of nodes. For our use case, this corresponds to structural changes due to non-stationary data. For example, such a change could be introduced by a software update changing the name of an executable. However, it does not reflect changes due to nondeterminism.

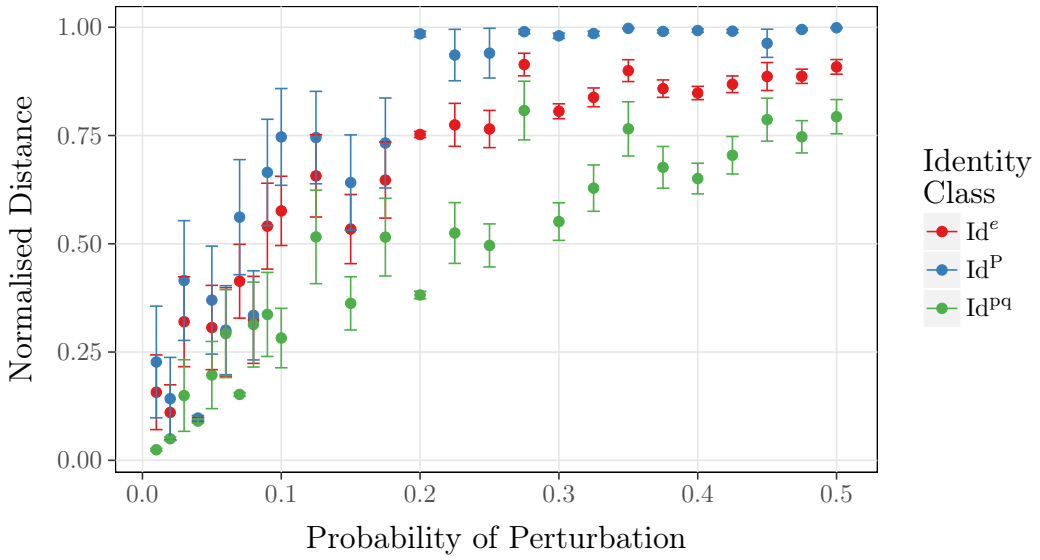


Figure 6.3.: Visualisation of an identity ensemble for partial similarity. Each identity uses a different context, allowing them to match different patterns individually or combined. We introduce perturbations to an existing tree by renaming vertices, which is equivalent of replacing the vertex.

Permutations of Consecutive Elements

Due to the nature of our use case, analysed trees are inherently nondeterministic at small scales. Each pilot and payload consists of multiple processes, some of which are executing concurrently. If multiple processes start or finish in a short time window, the sequence of processes across different threads of execution is not deterministic. This introduces *micro changes* in the order of siblings and concurrent processes. As a result, permutations can occur in the order of consecutive children in $\mathcal{Q}$ dimension or the order of events in $\mathcal{S}$ dimension.

In the following we consider micro changes to influence a limited neighbourhood of siblings within a tree T. A micro change is expressed by the permutation of two vertices $u, v \in \mathrm{V}(T)$. For further considerations we introduce a *width* that specifies the influence of micro changes within a tree T. The *width* specifies how far apart the two vertices u, v can be to each other, that is up to $width - 1$ vertices lie between u and v.

Furthermore we assume the chance of overlapping micro changes to be negligible. That means that two micro changes are practically never intertwined. Furthermore, a vertex that is already affected by a micro change cannot be part of another micro change.

To specify an identity ensemble that deals with permutations regarding micro changes of a given *width* we first consider the identity class $\mathrm{Id}_q^{\mathrm{Pq}}$ (see Section 4.4.4 on page 64 for further details). The given identity class defines $\mathcal{P}$ with infinite-length and $\mathcal{Q}$ with fixed-length encoding. Without utilisation of identity ensembles, one micro change of a given *width* changes up to $q + width$ identities. Consequently, the bigger the *width* of micro changes and the longer the sequence of vertices in $\mathcal{Q}$ dimension, the bigger the possible distortion in the resulting distance measure.

To lessen this impact, we adapt the approach used by windowed pq-grams for unordered trees [25]. Instead of using a nondeterministic ordering, we sort vertices in a dimension

Algorithm 5 Identification of vertices for $\mathcal{Q}$ and $\mathcal{V}$ dimension

Precondition:
`v` is the anchor vertex
`q` is number of elements in $\mathcal{Q}$ dimension
`w` is the *width* of micro changes
Postcondition: $\mathcal{Q}$ and $\mathcal{V}$ dimension

```
1: function buildIdentity(v, q, w)
2:     nodes ← empty list
3:     append v to nodes
4:     i ← 1
5:     while i < q + w − 1 do
6:         append i'th order sibling of v to nodes
7:         i ← i + 1
8:     sort nodes in descending order
9:     V ← |nodes[0]⟩
10:    Q ← |nodes[1], ..., nodes[q]⟩
```

to guarantee a deterministic ordering. In contrast to windowed pq-grams, which sort all children of a vertex and select a window from the result, we first select vertices within a window and sort the resulting sample.

To accommodate the distance distortion caused by micro changes of a given *width* we therefore introduce an identity ensemble to complement the identity class $\mathrm{Id}_q^{\mathrm{Pq}}$. We therefore introduce a further identity class in $\mathcal{Q}$ and $\mathcal{P}$ dimensions that is defined dependent on the former identity class $\mathrm{Id}_q^{\mathrm{Pq}}$, that is

$$\mathrm{Id}_{q,w}^{\mathrm{PqOrder}} = |\mathcal{P}_\infty, \mathcal{Q}_{q,w}, \mathcal{S} = \varnothing, \mathcal{V}_{q,w}\rangle.$$

It mimics the same definition as $\mathrm{Id}_q^{\mathrm{Pq}}$ in $\mathcal{P}$ dimension but extends the handling of the $\mathcal{Q}$ dimension. The algorithm to build the identity for $\mathcal{Q}$ and $\mathcal{V}$ dimension is given in Algorithm 5. Depending on the length of $\mathcal{Q}$ dimension of $\mathrm{Id}_q^{\mathrm{Pq}}$ and a given *width* of micro changes the algorithm determines the number of eligible vertices for $\mathcal{V}$ and $\mathcal{Q}$ dimension. Then, the vertices are sorted in descending order. $\mathcal{V}$ is assigned the last vertex while the following q vertices are stored in the $\mathcal{Q}$ dimension.

The algorithm uses a window of $q + width$ vertices to ensure that the complete pair of potentially permuted vertices is selected. Furthermore, to ensure comparable sensitivity as the underlying identity class, only q vertices are considered for the final identity in $\mathcal{Q}$ dimension.

Figure 6.4 on the facing page visualises the distance distortion for a growing amount of micro changes for a given *width* of 1. The identity considers an extent of $q = 2$ in $\mathcal{Q}$. The Figure visualises the reduction of effects from micro changes with an independent encoding of identity ensemble $\mathrm{Id}_{q,w}^{\mathrm{PqOrder}}$. The strict $\mathrm{Id}_q^{\mathrm{Pq}}$ is strongly affected by permutations, as a permutation affects every enclosing window in $\mathcal{Q}$. In contrast, the $\mathrm{Id}_{q,w}^{\mathrm{PqOrder}}$ fails to match only if the permutation is at one end of the window in $\mathcal{Q}$. However, as signified by the smaller spread, the $\mathrm{Id}_{q,w}^{\mathrm{PqOrder}}$ fails to distinguish some constellations. For example, it can hide permutations if the fanout is low. The identity ensemble retains features of both identities.

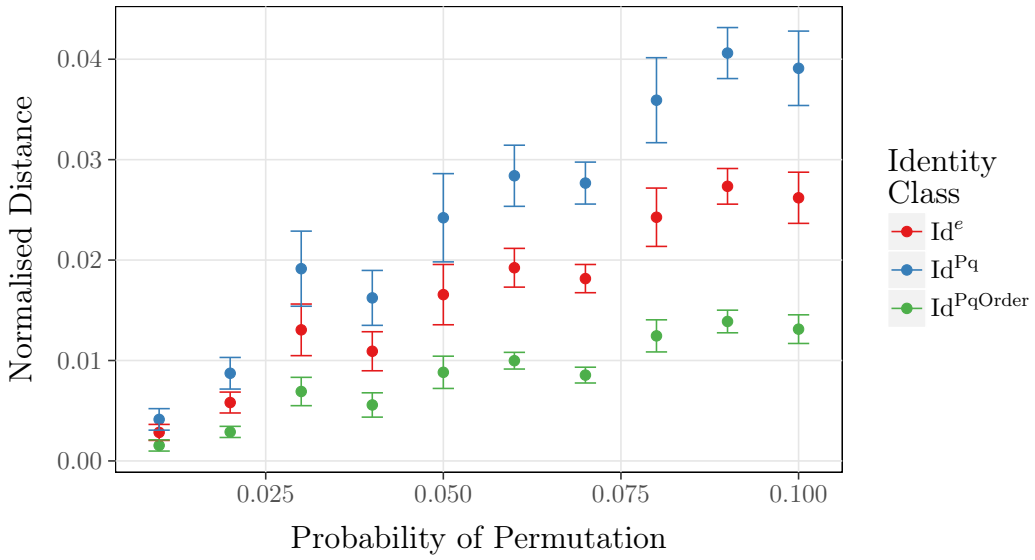


Figure 6.4.: Visualisation of an identity ensemble that handles micro changes. Both identities use $q = 2$ in $\mathcal{Q}$, and a permutation width $w = 1$ is used. The effective fraction of permutations is notably lower than the permutation probability. This results from permutations of identical siblings.

6.4. Summary

Tree distances can be defined with a variety of models, which are adequate for different use cases. Both literature and our own work suggest several models suitable for our use case in general. The black box nature of our use case makes it unfeasible to select a single, strict model. Instead, it is desirable to use multiple distance models in parallel to cover multiple possible features.

We propose a method to express multiple identity classes as a single identity ensemble. This computes and evaluates multiple identities for every vertex, which can be resolved independently or dependently as desired. For our work, we choose an independent resolution and aggregation of identities. This reflects the inherent uncertainty on how individual features in a black-box use case interact, and is the most robust for stream processing.

The introduction of identity ensembles allows for auxiliary identities in addition to previously defined identities. These auxiliary identities are less restrictive, allowing for partial matches across trees. In combination, an identity ensemble of strict and auxiliary identities can express a range of similarities and differences, while retaining $\mathcal{O}(1)$ time complexity per vertex.

In specific, we have introduced an auxiliary identity that absorbs local permutations in the order of vertices. This reflects our use case, where the real execution environment implies nondeterminism of order at small scales. As such, the introduction of identity ensembles and auxiliary identities enables robust distance approximation for dynamic trees that are unordered at small scale, and ordered at large scale.

7. Online Analysis of Dynamic Streaming Trees

The distance measures introduced so far provide the basis for online analyses of dynamic trees. As our formalisation explicitly defines an incremental variant, introduced measures can be directly applied to streaming data. We have introduced extensions with regard to preserving the complexity of our simplest approach. However, monitoring jobs requires the analysis of thousands of jobs in parallel.

Within this Chapter, we introduce a pipeline for processing event streams of dynamic trees. This pipeline connects the event streams from our monitoring, the incremental distance measurements of trees and attributes, and the superposition of elements. The target of this pipeline is the online distance measurement as an enabler for subsequent online clustering and classification.

In the following, we explore the basic approach to processing a high number of monitoring streams in parallel. The focus is on how the incremental behaviour of distance measuring can be preserved in clustering. In specific, we explore how various features of our distance measure allow for an efficient comparison of many trees at once.

Some of the concepts of online analysis regarding the clustering of dynamic trees to detect anomalies have originally been published in Kuehn [137].

7.1. Related Work

A variety of data mining approaches for analysis of semi-structured data have been designed. Most approaches in literature focus on clustering and classification of documents such as XML. Especially hierarchical clustering approaches have been largely adopted due to high quality of results [71, 73, 148, 151, 167].

Most of those methods are adaptions of agglomerative hierarchical clustering algorithms. This means, they implement a bottom-up strategy where each single tree is considered to be a cluster first. The clustering iteratively merges least dissimilar clusters until an optimal partitioning with regard to a pre-defined quality measure is reached. However, hierarchical clustering approaches are quadratic in time and thus are not scalable and cannot be considered for online analysis.

The methods focus on different decomposition strategies. In Nierman and Jagadish [167], for example, the authors utilise the TED to measure distance between two documents. This is due to the complexity of TED being rather expensive. Other approaches therefore improve the method by representing trees by other data structures that retain the structure of trees [9, 74, 88, 115, 151, 165]. All of these methods focus on structural features of trees and neglect content information.

Some methods explicitly focus on integration of content and structure [37, 98, 223, 236]. But there is still a lack for efficient methods that combine structure and content for successive analysis. This is mainly due to the sheer size and complexity of using elements to describe both features [143]. This is especially true in the era of big data analysis as well

as dynamic data analysis in an online manner. Especially the consideration of dynamic data requires deliberate representation of data to prevent concentration of distance values for high-dimensional data [56]. This problem results in a lack of separability within the clustering and should, therefore, be avoided [38]. Thus, only meaningful data must be considered for representation. Consequently, it is essential to combine content features and structural features to derive meaningful clustering results.

In summary, it can be stated, that most methods focus on clustering or classification of static trees. However, we need to consider dynamic trees and therefore strive for an incremental clustering that is robust to the continuous changes of trees.

7.1.1. Selection of Algorithm

To realise an incremental clustering of streaming dynamic trees, we strive for an approach that complies with the following requirements: Due to continuous changes to the objects being clustered, the clustering approach itself needs to be incremental. However, it should also be robust. This means, the incremental change of a dynamic tree that is part of the clustering must not constantly change the identified clusters. Based on considerations regarding our use case to anomaly detection of batch jobs (compare Section 2.2.3 on page 19) we also require a clustering approach that is robust to outliers and enables their detection. Furthermore, HEP batch jobs are built from a few common frameworks. For example, batch jobs from the same collaboration can differ in details we must be sensitive to, but still, share the same fundamental architecture. We therefore strive for an approach supporting overlapping clusters. Finally, the approach should support scalability concerning space and time complexity.

Based on different models and definitions of a cluster existing clustering algorithms can be distinguished. As we represent relationship for two dynamic trees by their distance we focus on distance-based clustering methods in the following before briefly describing our selected clustering approach. We, therefore, distinguish connectivity-based, centroid-based, and density-based clustering.

Connectivity-based clustering also known as hierarchical clustering is based on the intuition that objects are more related to nearby objects. Hierarchical algorithms therefore iteratively connect objects that are closest to each other based on their distance until all objects are connected. Hierarchical clustering does not produce a single partitioning of the data but a complete hierarchy of partitionings. However, due to the iterative procedure time complexity of hierarchical clustering is $\mathcal{O}(n^2)$. Furthermore the approach lacks robustness as hierarchical clustering is sensitive to noise and outliers.

Centroid-based clustering algorithms such as k-means [232] find the k cluster centres and assign each object to its nearest cluster centre, such that the squared distances between objects and cluster centres are minimised. Usually centroid-based clustering algorithms rely on the input parameter k to define a fixed number of clusters [104]. However, also when the number of clusters is known in advance, the algorithm only finds a local optimum. Furthermore it is sensitive to noise and outliers.

Density-based clustering algorithms define clusters as regions of high object density. Objects in sparse areas are considered noise. Thus, density-based clustering by design differentiates noise from clusters. As such, the clustering is stable even in the presence of outliers. The most popular density-based clustering is DBSCAN [85]. However, in the following we will focus on DenGraph that is an extension of DBSCAN [86, 192]. DenGraph is a density-based graph clustering algorithm with a time complexity of $\mathcal{O}(n)$ for a number

of n objects that are clustered. Notably, DenGraph allows the borders of clusters to overlap, representing similar but distinct clusters. Furthermore, this makes clustering deterministic regardless of ordering, as required for streams. In addition, DenGraph supports incremental processing of nodes and edges as well as a recursive generation of sub-clusters to obtain a hierarchy of clusters [192].

7.2. Overview of the Approach

The distance measurement framework for dynamic tree (see Chapter 4 on page 39) and its extensions for attributes (see Chapter 5 on page 77) as well as ensemble-based distances (see Chapter 6 on page 101) provides the means to handle streaming dynamic trees from our monitoring sensors (see Chapter 3 on page 23). The goal is to evaluate the dynamic trees as they are streamed in, with a focus on identifying groups and similarly outliers. In the following Sections, we focus on the technical aspect of this: We require a means to efficiently apply our incremental distance measurement to cluster monitoring data. This forms the technical framework on which we then evaluate the criteria to meaningfully reason about jobs (see Chapter 8 on page 129).

As we are working with multiple streams each describing a distinct dynamic tree, we propose an approach using separate processing pipelines. An overview of the concept is shown in Figure 7.1 on the next page. The input into the *vertex identity pipeline* is divided into two separate streams. The first stream follows an event-based concept and describes the evolving structure of a dynamic tree with start and end events of vertices. The second stream is based on a sampling model and provides attribute events at discrete intervals. This distinction into event-based and sampling-based event streams originates in our monitoring use case. Our formalism is not negatively affected by this distinction. The formalism itself is defined on any frequency of events, both equally-spaced or irregularly-spaced events. The output of the vertex identity pipeline is a stream itself again. The pipeline focuses on a non-blocking processing of input events to produce the incremental distance events. Thus, the pipeline concept enables flexibility and more importantly online analysis of distance results for dynamic trees.

Whenever an event is processed within the pipeline, first its vertex identity is generated unless it is already known. This identity is supplemented with a number of distribution statistics defined by the choice of identity profile projection operators. The combination of vertex identity and its distributions forms the identity for the given event. However, based on the given use case several identity classes can be combined to an identity ensemble. This identity ensemble or individual identity is considered for distance calculation.

The distance calculation itself relies on the input of two objects. Within our framework we enable different combinations of objects that are considered for distance calculation:

- distance calculation between two stored identity profiles or identity ensemble profiles[1] from disk,
- distance calculation for a streaming dynamic tree and a stored identity profile, and
- distance calculation for two streaming dynamic trees.

[1] There is no distinction between identity profiles and identity ensemble profiles in our proposed formalism. An identity ensemble profile can be considered a generalisation of identity profiles. To simplify further discussions we do not distinguish between identity profiles and identity ensemble profiles. Instead, we only refer to identity profiles.

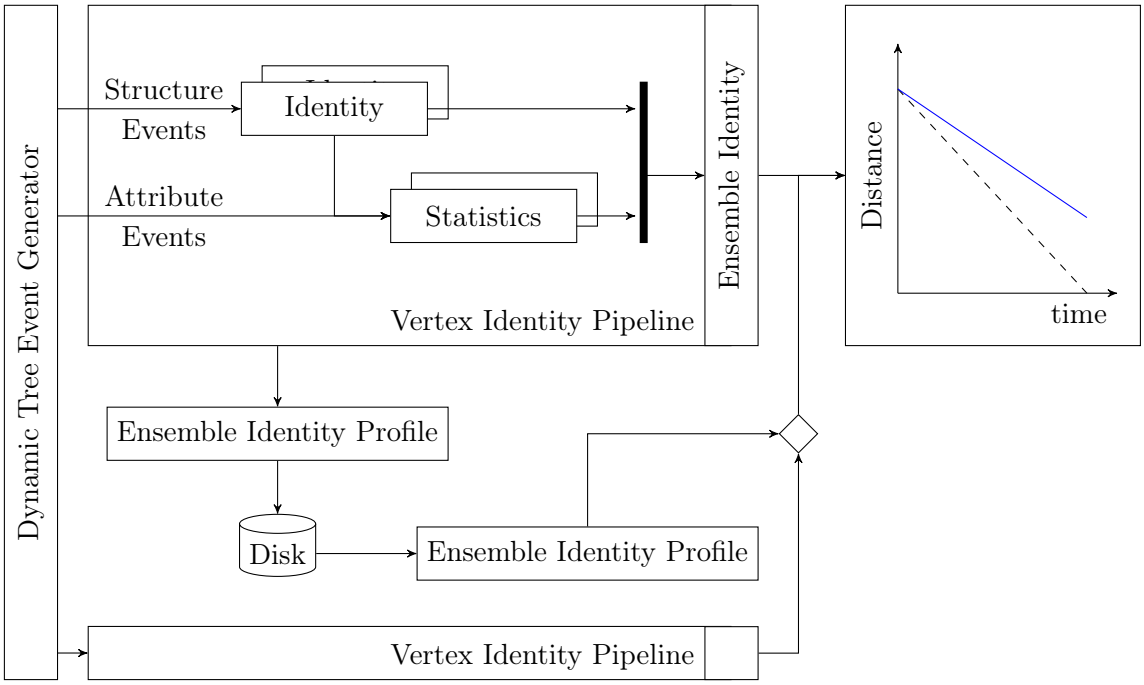


Figure 7.1.: Structure of dynamic tree distance measurement framework. The framework implements a pipelining concept that ensures that the output is a stream itself. The input to the pipeline are two streams that contain events modifying the structure of a tree as well as events describing attribute changes. For each event a vertex identity is generated before related distribution statistics are assigned. Distance calculation is either performed on an individual identity or a combination of different identities, that is an identity ensemble. In addition, the distance calculation component requires a second tree to compare the stream to. The framework supports two different inputs: another streaming dynamic tree or a recorded identity ensemble profile from disk.

Furthermore we want to stress, that we support tree event stream representation for static trees, enabling distance calculation between static and dynamic trees.

For an online analysis, distance calculation between either two dynamic streaming trees or one dynamic streaming tree and a recorded identity profile are reasonable. However, distance calculation between two dynamic streaming trees does not have the characteristic of monotonicity. This characteristic is especially relevant for outlier detection. Additionally, while we expect non-stationary data, our analysis shows that data is practically stationary well over the runtime of a single job. Thus, using stored identity profiles allows us to compare currently running jobs against a wider range of groups of homogeneous jobs. In the following, we therefore restrict the distance measurement to distance calculations between a streaming dynamic tree and a recorded identity profile.

7.3. Incremental Clustering

To realise an incremental clustering of dynamic trees we reviewed existing approaches (see Section 7.1 on page 113) that allow the clustering of semi-structured data. To the best of

our knowledge we are the first to introduce an incremental clustering of streaming dynamic trees.

As discussed previously, density-based clustering fulfils our basic requirements on clustering. From many available density-based clustering approaches we exemplarily select the density-based clustering approach DenGraph. Density-based clustering is robust against outliers, and naturally represents outliers. Both insertions and deletions are well-behaved, making density-based clustering suitable for incremental changes in principle. Furthermore, several density-based clustering approaches support hierarchical clustering, allowing us to inspect further details in clusters. Finally, density-based clustering usually works on arbitrary undirected graphs, and does not rely on geometric properties of an Euclidian space – this is a critical feature for our distances derived from deeply nested data structures.

7.3.1. DenGraph

Without loss of generality, we have chosen the DenGraph algorithm for our work. While extensions of the algorithm are optimised to handle partial and incremental graphs, the general working principle is the same. The DenGraph algorithms cluster nodes of a graph $G = (V, E)$ consisting of a set of nodes $\mathrm{V}(G)$ and a set of weighted, undirected edges $\mathrm{E}(G)$. The weights correspond to the distance between any two nodes that are connected by an edge.

The core concept of the algorithm is the *neighbourhood of a node*. This neighbourhood is evaluated regarding two parameters: The parameter ϵ defines the maximum distance to a connected node. Any two nodes u, v are treated as neighbours when their corresponding edge weight is smaller or equal to ϵ.

The ϵ neighbourhood of a node $v \in V(G)$ is given by

$$\mathrm{N}^{\epsilon}(v) = \{u \,|\, \mathrm{dist}(u, v) \leq \epsilon, \forall u \in \mathrm{V}(G)\}, \tag{7.1}$$

where $\mathrm{dist}(u, v)$ is the distance between the two nodes u and v.

The second parameter η defines the number of neighbours that are required to categorise a node to grow a new cluster. Each such node is called a *core node* and is defined by

$$\mathrm{V}^{\mathrm{core}} = \{v \,|\, |\mathrm{N}^{\epsilon}(v)| \geq \eta\}, \forall v \in \mathrm{V}(G). \tag{7.2}$$

Core nodes and their neighbourhood define the DenGraph clusters. The concept of the neighbourhood of nodes is visualised in Figure 7.2 on the next page. Border nodes are not core nodes, but lie in the neighbourhood of a core node. Any node not in the neighbourhood of a core node is considered noise.

The strict limitation of distances means that DenGraph operates on a limited number of edges within the graph G. This limitation is defined by the parameters ϵ and η specifying the region of interest. Thus, DenGraph effectively operates on sparse graphs, even if the initial graph is complete. On the one hand, this allows for $\mathcal{O}(n)$ in non-degenerate graphs. On the other hand, it means changing any individual node has a well-defined scope of effect.

Because of the dynamic behaviour of our data and the goal to achieve an online analysis, a repeated static clustering is inefficient. The dynamic data and potential evolution of clusters over time are best handled via incremental clustering. Thus, we use the DenGraph-IO [192] algorithm, which supports incremental insertion, deletion, and modification of nodes and edges in an existing clustering.

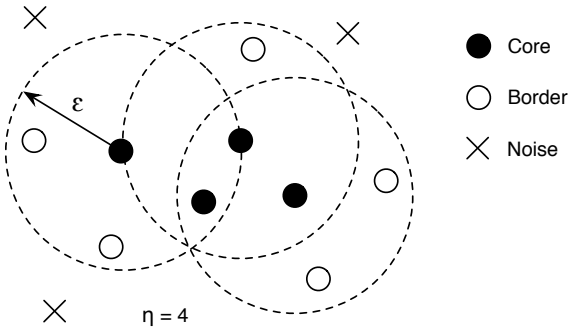


Figure 7.2.: Visualisation of the neighbourhood of nodes in density-based clustering approaches. Density-based clustering approaches differentiate core, border, and noise nodes. The categorisation of nodes is determined by evaluating a required number η of nodes within a distance of ϵ. A node that has at least η nodes in its ϵ neighbourhood is considered a *core* node. Nodes that are no core nodes but have at least one core node in its ϵ neighbourhood are so-called *border* nodes. Remaining nodes that are neither core nor border nodes are considered *noise*.

7.3.2. Clustering Dynamic Trees

The challenge of clustering dynamic trees maps to the tree-to-tree similarity search for dynamic trees. Given a dynamic tree $\mathscr{T}$ and a set of completed trees $\Gamma = \{\mathscr{T}_1, \ldots, \mathscr{T}_m\}$, find the trees $\mathscr{T}_i$ most similar to $\mathscr{T}$. Based on our proposed framework to distance measurement for dynamic trees we can consider the set Γ as a set of identity profiles that can be utilised for distance calculation, that is $\Gamma = \{\langle\mathscr{T}_1|, \ldots, \langle\mathscr{T}_m|\}$. We further assume this set Γ to be the basis for an initial clustering to enable incremental clustering of dynamic trees.

Distances on Γ are defined by $\text{dist}(\mathscr{T}_i, \mathscr{T}_j), \forall i, j < |\Gamma|$ as previously introduced (see Chapters 4 on page 39 and 6 on page 101). By utilising DenGraph, we cluster our set of existing dynamic trees Γ and assume to derive a valid clustering with a set of k clusters, that is $C = \{C_1, \ldots, C_k\}$ with $|C| = k$.

However, performing the initial clustering requires $\mathcal{O}(m^2)$ distance calculations for m tree identity profiles as we do not have sparse edges between the individual tree identity profiles. While DenGraph operates on a weighted graph in $\mathcal{O}(m)$, our use case requires the calculation of *all* edges to test $\text{dist}(\mathscr{T}_i, \mathscr{T}_j) \leq \epsilon$ for pruning. Thus, we need to consider a fully meshed graph, requiring $\mathcal{O}(m^2)$ distance calculations. When incrementally inserting dynamic trees into the existing clustering it still requires $\mathcal{O}(m)$ distance calculations per tree to consider. Thus, we focus on the minimisation of search space in the following. Otherwise scalability for the proposed online clustering cannot be guaranteed.

Sampling

Our approach of identifying groups of jobs hinges on the assumption that jobs of the same workflow are interchangeable with some level of uncertainty. In turn, this means that given a sufficient sample of jobs, additional jobs of the same workflow have negligible differences to those of the sample.

This redundancy of information allows a reduction of the search space for clustering. Instead of storing all monitored jobs, it is sufficient to work with a subset of data [188, 190].

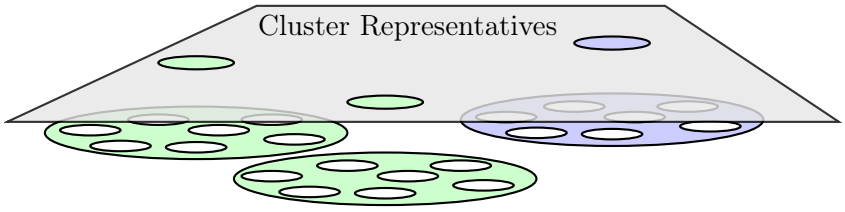


Figure 7.3.: Overview of concept of Cluster Representatives. To efficiently represent dense clusters, CRs act as a replacement for groups of nodes. Each CR represents its group of nodes with a given accuracy. For arbitrarily-shaped clusters, several CRs can be used to represent individual regions of the cluster. As such, large clusters may be represented by multiple CRs as once.

Thus, we can reduce the number of identity profiles that must be considered in distance calculations.

Density-based clustering allows to define a minimum information density sufficient to describe clusters. This means that for a distance uncertainty $\epsilon_\rho \leq \epsilon$ we preserve no more than a number of $\eta_\rho \geq \eta\,(\epsilon/\epsilon_\rho)^d$ data points, for an estimate of dimensionality d. An estimation for the dimensionality d must be derived from features of a given set of data Γ. Thus, the estimation is use case specific and requires a repeated validation for non-stationary data.

Downsampling further allows to reflect that data is inherently non-stationary. As monitoring constantly streams in new data, the volume of recorded data is theoretically unbounded. However, only recent data can be considered an indicator for the current normal behaviour.

Instead of using the entire history of recorded data for clustering, only recent data is required. As we use an incremental clustering, it is feasible to remove existing data. This allows us to bias and eventually remove old data. With clustering reflecting this incrementally, existing clusters naturally drift towards recent behaviour. This also includes changes where clusters disappear over time and are replaced by new clusters.

Although sampling allows a significant reduction of tree identity profiles for dense clusters, distance calculations are still required for m_ρ tree identity profiles, with $m_\rho \leq m$. We therefore consider the aggregation of tree identity profiles to represent distinct clusters.

Cluster Representatives

To improve the complexity of distance calculations, clustering methods introduce the concept of *Cluster Representatives (CRs)* [58, 70, 71, 190]. A CR describes common features of all members of a given cluster. Thus, a cluster can be depicted equivalently by its members or its CR with some well-defined accuracy. As such, a CR can be used in place of the members of its cluster for many operations.

In [37] the authors introduce CRs as a union of documents. However, the given approach does not address different multiplicities of elements inside documents. Instead, we require an aggregation of the identity profiles of trees contained in the group represented by a CR.

Since multiplicity is defined via attributes (see Section 5.3.1 on page 86), aggregation can be expressed by the merging of attributes. For multiplicities, this merging corresponds to the average multiplicity of each element in the group. In general, the merging of attributes minimises the difference between the CR and all members of its group.

Definition 7.1 (Aggregated identity profile)**.** Let Γ be a collection of identity profiles, that is $\Gamma = \{\{|\mathscr{T}_1\rangle, \ldots, |\mathscr{T}_g\rangle\}\}$ and θ a set of identity profile projection operators. An *aggregated identity profile* is the mean identity profile $|\mathscr{T}'\rangle$ maximising the similarity to the identity profiles of Γ by with respect to attribute statistics:

$$\langle \mathscr{T}'|\theta|\Gamma\rangle = \sum_k \langle \mathscr{T}'|\theta|\mathscr{T}_k\rangle. \tag{7.3}$$

For clusters of arbitrary shape, a single CR is not necessarily capable of describing all members of a cluster within a given accuracy. This is especially the case in non-Euclidian space, where the concept of averages is meaningless on a large scale. Thus, we use a set of CRs for each cluster, each representing a different region of the cluster (see Figure 7.3 on the previous page). Each CR unifies a set of tree objects for the given region. Hence, the number of cluster representatives is always smaller or equal than the number of objects belonging to their cluster.

Definition 7.2 (Cluster Representative (CR))**.** Let C be a cluster consisting of a set of identity profiles, that is $C = \{|\mathscr{T}_1\rangle, \ldots, |\mathscr{T}_m\rangle\}$, and θ a set of identity profile projection operators. The set of *Cluster Representative (CR)* of C, denoted by $C^{\mathrm{CR}} = \{|c_1\rangle, \ldots, |c_r\rangle\}$, is the set of *aggregated* identity profiles that optimises the overlap between identity profiles and aggregated identity profiles:

$$\langle C^{\mathrm{CR}}|\theta|C\rangle = \sum_r \sum_m \langle c_r|\theta|\mathscr{T}_m\rangle. \tag{7.4}$$

Optimal overlap minimises the number of CRs r, and approaches a desired accuracy ϑ:

$$\frac{\langle C^{\mathrm{CR}}|\theta|C\rangle}{\langle C|\theta|C\rangle} \to \vartheta. \tag{7.5}$$

This criteria implicitly satisfies several features desirable for identity profiles: Minimising the number of CRs k within a given accuracy avoids using the nodes directly as CRs. The general maximisation of overlap ensures that the cluster is adequately split into groups, as a single CR representing several groups degrades overlap. Finally, a target accuracy ϑ corresponds to clustering based on an ϵ neighbourhood; for $\vartheta = 1 - \epsilon$, every CR represents an ϵ neighbourhood.

The formation of CRs follows directly from the given choice of identity class and identity profile projection operator. We profit from our proposed framework given in Chapters 4 on page 39 and 5 on page 77 as no further concept nor definition needs to be introduced to realise the aggregation of CRs. In turn, this gives CRs other properties of our approach, such as supporting an incremental creation (see Algorithm 6 on the facing page).

The use of CRs reduces the number of existing nodes that new nodes must be compared against. However, this must still be done explicitly for every CR. For a given cluster, it is reasonable to assume that CRs share many identities. To simplify the association to specific trees or clusters given an identity we further utilise the concept of inverted indexes [66]. An inverted index maps each identity to the set of trees or clusters containing this identity. An inverted index for each cluster means that identities are not compared to $\mathcal{O}(kr)$ CRs but only $\mathcal{O}(k)$ inverted indices of clusters.

This approach to represent clusters by CRs has several advantages: First, we can represent arbitrary-shaped clusters. Second, we unify several overlapping identities and thus can perform direct lookups. This direct lookup avoids linear search over a number of trees to determine distances. Third we improve statistical reliability by unifying homogeneous identity profiles.

Algorithm 6 Incremental creation of Cluster Representatives

Precondition:
`C` is the current cluster to consider
$|\mathscr{T}\rangle$ is the identity profile to be inserted to `C`
ω is the distance threshold for two identity profiles
Postcondition: updated cluster `C` containing current CRs

```
1: function AddProfile(|𝒯⟩, C)
2:     distances ← array of length |C^CR| initialised with 0
3:     for identity ∈ |𝒯⟩ do
4:         distance ← distance(C, identity)
5:         distances ← update distances element-wise with current distance
6:     if min(distances) < ω then
7:         update CR with smallest distance by |𝒯⟩
8:     else
9:         add new CR to C and add |𝒯⟩
10:    return C
```

7.4. Incremental Classification of Dynamic Trees

In contrast to clustering and simplifying recorded jobs, the classification of monitored jobs must be performed online to be meaningful. The duration of each job suggests that a latency for decisions in the order of seconds is acceptable (for details on the distribution of duration for CMS jobs see Figure C.3 on page 178). However, classifying several thousand jobs in parallel requires an efficient approach. As such, we exploit features of both clustering and distance measurement to reduce complexity.

7.4.1. Dynamic Probing with Virtual Nodes

Our choice of density-based clustering is already motivated by reducing the complexity of identifying clusters. Thus, the existing clustering mechanism provides key features for an efficient classification of nodes to clusters. Most importantly, the density-based clustering limits comparisons to a strictly-defined neighbourhood. Additionally, incremental clustering offers an existing, efficient way of representing dynamic distances.

However, inserting several thousands of vertices is likely to influence the clustering itself. The integration of a dynamic tree $\mathscr{T}$ as a node into a clustering built from finished dynamic trees would skew clusters due to incomplete data of $\mathscr{T}$. However, categorising the tree $\mathscr{T}$ at runtime is considered important for responsiveness and optimisation of time and space complexity.

We, therefore, exploit clustering for classification by introducing the concept of *virtual nodes*. The general concept of virtual nodes is visualised in Figure 7.4 on the following page. A virtual node is tightly bound to the underlying clustering mechanism, without being part of the clustered nodes. However, it is still part of the clustering and thus profits from incremental changes to the clustering: Distances to relevant clusters can be updated incrementally.

To not skew the actual clustering while still classifying incomplete dynamic trees, we extend DenGraph with a probing mechanism based on the concept of virtual nodes. In

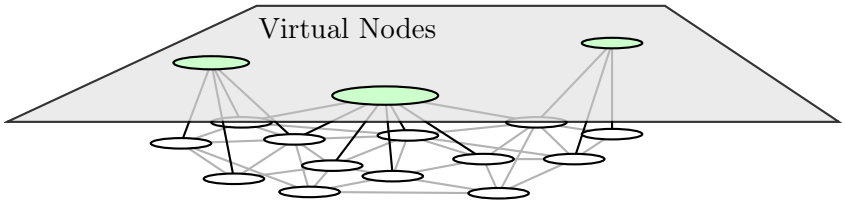


Figure 7.4.: Dynamic data probing with virtual nodes. A virtual node is not considered in the proper clustering process but is part of the overall mechanism. By exploiting virtual nodes the clustering of dynamic objects is enabled. Each virtual node encapsulates an objects dynamics and keeps the required incremental distances based on the underlying clustering. Whenever a virtual node is finished, it can directly be added to the proper clustering without any recalculation of distances.

addition to the nodes forming existing clusters, we add a conceptual layer on top. Nodes corresponding to currently monitored dynamic trees are inserted into this layer representing the virtual nodes.

This overlay works similar to the regular layer of nodes. Most importantly, the ϵ neighbourhood of the virtual nodes includes nodes in the regular layer. As such, these virtual nodes are automatically classified as noise, border, or core nodes.

Virtual nodes support the same classification as regular nodes: Virtual nodes can be border nodes of multiple clusters, and even form a bridge between clusters if they are core nodes of multiple clusters. Additionally, incremental updates of virtual nodes are efficiently evaluated in their ϵ neighbourhood.

However, there is no back propagation to the regular layer of nodes. Regular nodes are not influenced by virtual nodes in their ϵ neighbourhood. This includes virtual border and core nodes, which can never promote a regular node to a border or core node.

The use of virtual nodes lends itself well to analysing multiple streams in parallel. Since the underlying clustering is immutable to the virtual nodes, it does not need to be duplicated for each event stream but can be shared safely. Additionally, multiple virtual nodes can be classified in parallel.

7.4.2. Divergence of Distances

A simple model of virtual nodes can be realised using insertion and deletion of the respective state T_i of the dynamic tree $\mathscr{T}$. However, this requires a repeated calculation of the local ϵ neighbourhood. Instead, we reuse the same virtual node during the lifetime of its corresponding dynamic tree. This allows for optimisations based on incremental distance changes.

Notably, we have purposely defined the incremental dynamic tree distance and any extensions to provide monotonic behaviour compared to an ideal behaviour (see Section 4.5.4 on page 73, Section 5.3 on page 86, and Section 6.3.1 on page 107). In short, ideal behaviour means an incremental identity profile matches the recorded identity profile it is compared to; at any step, a dynamic tree can only be equal to or worse than the ideal behaviour.

Notably, this property is only guaranteed for our choice of comparing incremental identity profiles against *recorded* identity profiles (see Section 7.2 on page 115). When comparing two incremental identity profiles, matching identities can be delayed for one stream. This

causes temporary distance when an identity occurs in one stream, which is compensated once the identity arrives in the other stream.

To optimise space and time requirements for classification, we exploit the monotonicity of the underlying distance measurement. Once a dynamic tree diverges from the ideal behaviour, it can never converge again.

Definition 7.3 (Divergent incremental dynamic tree distance)**.** The recurrence formula for the *divergent incremental dynamic tree distance* between a recorded identity profile of a dynamic tree $\mathscr{T}_1$ and an observed dynamic tree $\mathscr{T}_2$ is given by

$$\begin{aligned} &\mathrm{dist}_0 = 0 \\ &\mathrm{dist}_i = \mathrm{dist}_{i-1} + \sum_j 2\alpha_j(1 - \langle \mathscr{T}_1 | \theta_j | P_{2,i}, Q_{2,i}, S_{2,i}, V_{2,i} \rangle), \text{ with } \sum_j \alpha_j = 1, \end{aligned} \tag{7.6}$$

where θ_j defines an identity profile projection operator (see Section 4.5.4 on page 73).

Compared to the regular *incremental dynamic tree distance*, this formulation requires an additional step if $|\mathscr{T}_1| > |\mathscr{T}_2|$ to compensate for missing vertices. However, the two are linked by a fixed, linear relation. It is inexpensive to incrementally calculate the regular and divergent distances from the same identity profile projection.

Notably, both distances have the same end result. However, the divergent distance is advantageous for incremental classification. The divergence of distances means that nodes can only leave the local ϵ neighbourhood of a virtual node, but they cannot enter it. Thus, by using the same virtual node for a dynamic tree, once a node diverges from the ϵ neighbourhood at any step, it does not need to be considered in further steps.

7.4.3. Convergence and Anomaly Detection

The divergent distances mean that candidate clusters are consecutively eliminated from the ϵ neighbourhood of a virtual node. This is the primary means of reducing complexity in our approach: for each dynamic tree, dozens of CRs must be compared in the first steps (see Section 7.3.2 on page 119). However, once the tree unfolds to regions where CRs diverge from a common trunk, many also exit the ϵ neighbourhood.

However, while the ϵ neighbourhood is based on relative distances, the divergent incremental dynamic tree distance is an absolute distance. The required normalisation of the relative dynamic tree distance (see Equation (4.27)) notably does not preserve monotonicity: the relative distance grows for mismatches, but shrinks for matches. Still, we can derive an estimate for a normalised distance that preserves divergence.

Proof. Let $\mathscr{T}_1$ and $\mathscr{T}_2$ be a recorded and dynamic tree, respectively. Let $|\mathscr{T}_1\rangle$ and $|\mathscr{T}_2\rangle$ be the respective identity profiles based on any identity and identity ensemble as well as identity profile projection operator θ. The relative dynamic tree distance at any stage i of $\mathscr{T}_2$ is given by Equation (4.27) as

$$\mathrm{dist}^{\mathrm{dynamic}}(\mathscr{T}_1, \mathscr{T}_{2,i}) = 1 - \frac{\langle \mathscr{T}_1 | \theta | \mathscr{T}_{2,i} \rangle}{\langle \mathscr{T}_1 | \theta | \mathscr{T}_1 \rangle + \langle \mathscr{T}_{2,i} | \theta | \mathscr{T}_{2,i} \rangle - \langle \mathscr{T}_1 | \theta | \mathscr{T}_{2,i} \rangle}$$

This distance is composed of the size of the recorded identity profile $\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle$, and the current overlap and difference.

$$= 1 - \frac{\overbrace{\langle \mathscr{T}_1|\theta|\mathscr{T}_{2,i}\rangle}^{\text{overlap}}}{\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle + \underbrace{\langle \mathscr{T}_{2,i}|\theta|\mathscr{T}_{2,i}\rangle - \langle \mathscr{T}_1|\theta|\mathscr{T}_{2,i}\rangle}_{\text{difference}}}$$

As an estimate, the overlap of identity profiles for $\mathscr{T}_1$ and $\mathscr{T}_2$ can never be greater than the identity profile of the known, recorded tree $\mathscr{T}_1$. Furthermore, the difference equals half the divergent incremental dynamic tree distance (see Equation (7.6)), which we denote by Ξ_i for brevity.

$$\geq 1 - \frac{\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle}{\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle + \Xi_i}$$

This provides an estimate for a normalised, divergent incremental dynamic tree distance measure. □

Notably, the precise distance lies within an ϵ neighbourhood if the monotonous, smaller estimate does as well. This allows us to define a simple threshold for convergence.

$$\begin{aligned} 1 - \frac{\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle}{\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle + \Xi_i} &\leq \epsilon \\ \frac{\Xi_i}{\langle \mathscr{T}_1|\theta|\mathscr{T}_1\rangle} &\leq \frac{\epsilon}{1-\epsilon} \end{aligned} \tag{7.7}$$

For dynamic trees conforming to a workflow, we expect the dynamic tree distance to converge to a cluster of that workflow within a threshold of ϵ. Notably, such trees must be tracked for their entire lifetime. Even an otherwise ideal tree can in principle diverge shortly before its expected end. However, we can expect a tree to diverge from clusters of other workflows, allowing for a reduction of comparisons. Table 7.1 on the next page shows the results for divergence of 1000 dynamic attributed trees with respect to 800 core nodes. 50 % of core nodes can be excluded prematurely from further distance calculations for unfinished dynamic trees.

Additionally, divergence offers an efficient tool to detect outliers. The dynamic tree of a job that deviates from known behaviour, for example due to software bugs, will also diverge in distance from all workflow clusters. In this case, no viable node remains in the ϵ neighbourhood of the virtual node. As these jobs do not correspond to normal behaviour by definition, we consider them anomalous.

7.4.4. Improvement of Anomaly Detection

To improve divergence and thus the detection point of anomalies while retaining accuracy we can exploit implications of vertex insertion events for dynamic trees. This observations relies on the requirement that after a vertex was added to a tree, it will be removed at a later point in time. Consequently, when the identity of a vertex $v \in \mathscr{T}$ does not match at insertion time, the identity of v will also not match at removal time. We can therefore anticipate the mismatching identity of the removal event during processing of the insertion event. Thus, the cost for a mismatching identity for insertion events is substituted by the sum of insertion and deletion for a mismatching vertex.

Table 7.1.: Dynamic tree distance divergence at runtime. Classification of dynamic trees is decided by association with the ϵ neighbourhood of cluster nodes. Dynamic trees that have finished can be precisely classified. Our approximation allows to detect the divergence of dynamic trees from a specific ϵ neighbourhood. Shown here are the final classification and runtime estimates for 1000 dynamic attributed trees with respect to 800 cluster nodes. On average, for every matching cluster node, 15 cluster nodes do not match. From these, our approximation excludes 50 % of candidates even without full knowledge of the dynamic tree. This is consistent with the fact that approximately half the dynamic trees are smaller than the recorded tree, thus not being targets of our approximation.

		Final	
		Divergent	Convergent
Runtime	Divergent	372 275	0
	Undecided	354 938	53 787

A sketch of the algorithm to improve the detection point of anomalies based on structural events is given in Algorithm 7 on the following page. For correctness, the mismatching deletion event in subsequent events is not counted again. This is ensured by a simple counter that validates the number of applied overestimates.

This technique results for our specific use case in an improvement of the anomaly detection point in 27.72 % of tree comparisons that appear anomalous with regard to a specified distance threshold. On average, the detection point occurs 15.20 % of events earlier without introducing any false positives.

7.5. Summary

This Chapter is dedicated to demonstrate how our distance measures can be combined to enable an efficient classification of our monitoring data. While our incremental measures are the foundation of an online analysis due to their low complexity, an approach to combine them for the end goal of classification is needed. Thus, we have proposed a pipelining approach which combines multiple parallel distance calculations with an incremental classification.

As our use case does not provide prior knowledge on classes, we have to derive them from our data. Since trees do not form an Euclidian space, we use the local density-based clustering approach of DenGraph. The class of density-based clustering algorithms is robust against outliers and explicitly identifies them as noise. Most importantly, each object only has an effect in a local scope, reducing complexity.

This clustering forms the basis for classifying trees described by our monitoring streams. The incremental distance calculation allows us to incrementally classify nodes by matching them to clusters. This concept is visualised in Figure 7.5 on page 127. To take full advantage of the existing clustering approach, we introduce virtual nodes: representing dynamic trees, they exploit the internal classification of the clustering for efficiency. However, to avoid skewing of clusters due to partial trees, regular nodes and clusters are not influenced by virtual nodes.

We use virtual nodes to optimise the classification of trees. By reformulating our

Algorithm 7 Improvement of divergence recognition for anomaly detection

Precondition:
$\sigma(\mathscr{T}')$ is the tree event stream representation of tree $\mathscr{T}'$
$|\mathscr{T}\rangle$ is the identity profile of the recorded tree

```
1:  function overestimateEvents(σ(T'), |T⟩)
2:      mismatches ← 0
3:      while event ∈ σ(T') do
4:          if type(event) = start event then
5:              if ⟨T|event⟩ = 0 then
6:                  overestimate distance
7:                  mismatches ← mismatches + 1
8:              else
9:                  handle distance as usual
10:         else if type(event) = end event then
11:             if ⟨T|event⟩ = 0 and mismatches > 0 then
12:                 skip distance handling
13:                 mismatches ← mismatches − 1
14:             else
15:                 handle distance as usual
```

incremental distances, we arrive at a monotonous measure for incremental dynamic tree distances. This allows us to discontinue comparisons to any tree which leaves the local neighbourhood at any time. Instead of continually comparing each monitored tree against all others, this limits comparisons to a converging window. If monitored trees are anomalous, this window becomes empty during the lifetime of a job, allowing for an early detection of outliers.

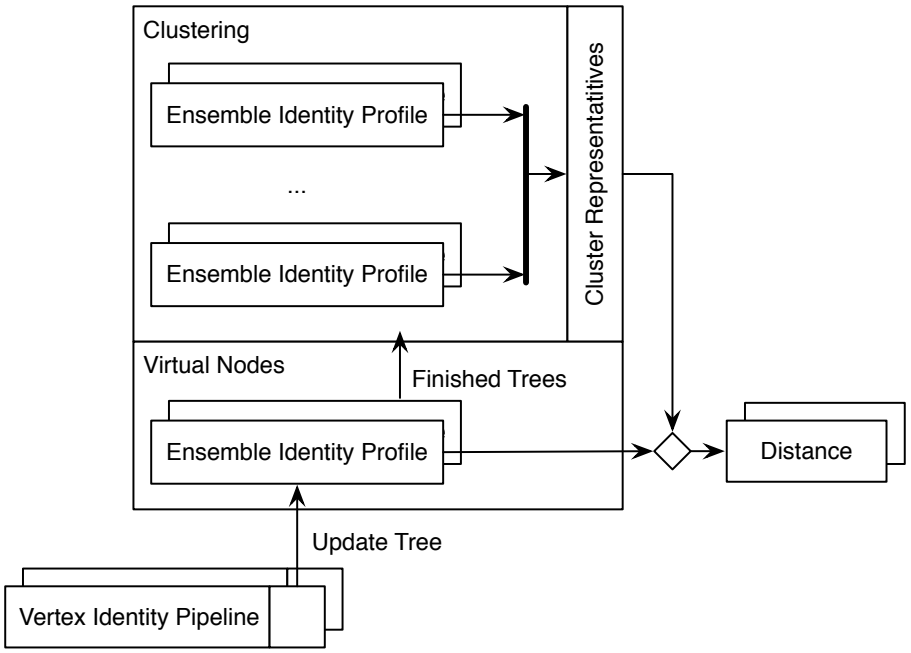


Figure 7.5.: Overview of utilisation of density-based clustering of dynamic trees to support online classification. For each event that is added by the vertex identity pipeline the associated virtual node is updated. For each update of a virtual node the distance with regard to current CRs of density-based clustering is updated. Whenever a tree is finished, the respective identity ensemble profile can be added to the clustering without recalculating all distances. A classification of trees is done based on the layer of virtual nodes. This classification is based on convergence to specific CRs. Incremental distance results enable the evaluation of convergence or event divergence to given CRs.

8. Evaluation

In the following, we evaluate the different parts of our proposed methodology for online analysis of dynamic trees in streaming environments. First, we start with an evaluation regarding relevant characteristics of our proposed dynamic tree distance measurement: scalability, sensitivity, and coverage, but also formal accuracy. We conclude the evaluation by presenting three different real-world use cases that reflect the entire workflow of online analysis of dynamic trees in streaming environments. In specific, we detail our findings for the analysis of HEP batch jobs regarding the ability to learn usage patterns, enable the utilisation of opportunistic resources, and towards an outlier detection during operation of the GridKa data and computing centre.

For reproducibility and better understanding all workflows we utilise for evaluation are documented in Appendix B on page 157. The workflows formally describe the data selection, data transformation, and data analyses processes that are exploited within this Chapter as well as previous Chapters. All results presented in this thesis rely on the results of these workflows.

To evaluate the proposed approach, we implemented a reference framework that is called Algorithm Simulation for Streaming Environments to aSsess tree Similarities (ASSESS) based on Python2.7. ASSESS implements all of the proposed methods to handle distance measurement of dynamic as well as static trees in streaming environments. For comparison with TED, we utilise the official Python module zss in version 1.1.2 [106]. The module zss implements the TED algorithm described in Zhang and Shasha [239]. As there is no official implementation of the DenGraph algorithm for Python, we use our own implementation published as the dengraph package [80]. For the DenGraph package we implemented the features described in Falkowski, Barth, and Spiliopoulou [86] and Schlitter, Falkowski, and Lässig [193] regarding the incremental update of clustering, the overlapping clusters as well as the creation of a hierarchy of clusters.

Some of the results presented in this Chapter have already been published in Kuehn and Streit [138].

8.1. Characteristics of Distance Measures

In Section 4.5 on page 67 we identified the characteristics of scalability, sensitivity, coverage as well as formal accuracy to be relevant for assessing the quality of dynamic tree distances. We, therefore, evaluate our proposed multi-step decomposition strategy for distance measurements with regard to these characteristics. Specifically, we consider the performance and characteristics using different, intuitive variants of both identity classes and distance functions.

As a basis for our evaluation, we first introduce the constraints, assumptions, and techniques behind our tests. They form the framework on which our evaluations are based on. In specific, this Section is dedicated to the study of our distance measurements under controlled conditions.

8.1.1. Conditions, Assumptions, and Techniques

As shown in Section 5.3.1 on page 96, incremental PDF statistics strictly expects Gaussian distributions, and deviates from the underlying distribution if it does not match. Our studies show that our data consists of attributes that do not follow a Gaussian distribution. To avoid introducing separate incremental PDF statistics for every attribute, we restrict ourselves to consider MultisetStatistics throughout this Chapter. We expect absolute differences for attribute values to become less significant for great value ranges. We, therefore, parameterise the MultisetStatistics with the transformation function $f(x) = \text{round}\,(\sqrt{x})$, which focuses on relative distances instead.

To systematically analyse the different characteristics of our distance measures, we require a methodology to control samples of dynamic trees and their distance to each other. Notably, we use recorded data only as a basis, on which modifications are applied in a controlled manner. These modifications allow us to evaluate our approach based on real data, but under well-defined conditions.

Present Dataset

The dataset that is utilised within this thesis is a sample recorded with the tool BPNetMon for user-centric monitoring introduced in Chapter 3.2 on page 28. Within a period of approximately one year the recordings of BPNetMon were stored as input for further analysis. The dataset contains batch jobs from 64 distinct worker nodes at the GridKa. Further information on hardware and setup used for the environment and monitoring can be found in Appendix D on page 179.

The original dataset contains the data from entire batch jobs, from any VO supported at GridKa. However, the evaluation presented here is performed using only batch jobs of a single VO.

- On the one hand, this simplifies the evaluation by ensuring limited reproducibility independent of our approach. Relying on the domain knowledge available for this VO, we can reliably obtain payloads and reason about their general, expected characteristics.
- On the other hand, this provides a realistic challenge for our approach. By limiting our data to a homogeneous setup, we avoid trivial distinctions from different technologies, and instead show the separability even for small differences.

Therefore, we have selected batch jobs of the CMS VO for further analysis. Table 8.1 on the next page summarises statistics about the dataset regarding the different batch jobs as well as the derived dataset of CMS payloads. Both datasets have also been analysed independent from available identity classes. Thus, the stated alphabet size is derived in terms of distinct Unix process names available in the dataset. Notably, both datasets provide a wide range of values for the different properties analysed. Especially the range of vertex counts requires a scalable algorithm.

Generation of Trees

We use data from a real world use case as the basis for our evaluation to portray the characteristics of our approach under realistic conditions. However, this means that we lack

Table 8.1.: Statistics on present datasets that are used for evaluation within the scope of this thesis. Statistics on pilots are considered for all 64 worker nodes we utilised for monitoring. The subset on payloads is collected from available pilots of 15 worker nodes. Payloads are only considered for the CMS experiment. Data for incomplete trees are skipped from our datasets.

Property	Pilots	CMS payloads
Samples	2 298 176	131 734
Duration (h)		
μ	8	3
max	420	60
vertices		
μ	8942	1706
max	3 880 692	3 639 289
Tree depth		
μ	15	8
max	37	22
Fanout		
μ	12	5
max	828	699
Alphabet size		
μ	95	46
max	458	379
Attributed vertices		
μ	31	9
max	9055	1503
Attribute events		
μ	466	106
max	111 744	6341

a robust basis for comparing our results against: An important motivation for our work is that the precise features of our data are not known in advance.

To benefit both from real data and from synthetic data, we use a hybrid approach. Instead of comparing arbitrary trees from our recorded data, we generate well-defined variants for each tree through selective modification. This includes standard edit operations akin to TED, such as renaming, deletion, and insertion of vertices but also a move operation. We also consider move operations here to replicate micro changes we experience in our original data. In addition, we also provide duplication, permutation, and repetition of entire branches. All of these operations can be controlled precisely by specifying the probability per operation. The operations can further be restricted to specific elements within the tree, for example leaf vertices, inner vertices, or vertices containing attributes.

To assess the characteristics of our approach, we empirically relate precise edit distances with a set of edit operations to our dynamic tree distance using tree decomposition based on identities. However, the computation of exact edit distance with moves is NP-hard [163]. TED for ordered trees still requires polynomial time [75, 209, 239]. Efficient approximations

have been proposed only for sorted trees in literature [22, 94, 233]. Even with a restriction to static trees, this is unfeasible given the complexity and scale of our data. To the best of our knowledge, there are no efficient algorithms to reliably approximate such edit distance, at least none which are not covered by our own approach.

We, therefore, exploit our generation mechanism, where a given dynamic or static tree $\mathscr{T}$ is subject to a series of random perturbations. As we perform these perturbations, we track the changes between the original tree $\mathscr{T}$ and the perturbed tree $\mathscr{T}'$. This is directly translated to equivalent edit distances, allowing us to derive an approximate tree edit distance at $\mathcal{O}(n)$ time complexity for generated trees[1]. To avoid any redundant operations, we perform perturbations in streaming order of the given tree. This enables us to compare the edit distance of a perturbed tree $\mathscr{T}'$ with our dynamic tree distance derived from the projection $\langle \mathscr{T} | \theta | \mathscr{T}' \rangle$.

Cost Models

Deriving the actual distance from individual edit operations is expressed with a cost model in TED. The classical TED assigns a uniform cost of 1 to any operation, such as the deletion of a vertex. However, other cost models are viable as well. In specific, our own view of process trees of jobs suggests a cost dependent on the size of the subtree rooted at the vertex under consideration.

The different cost models presented in the following are based on the standard edit operations of TED [75, 239]: deletion of a vertex, insertion of a new vertex as well as renaming of a label of a vertex (see Section 4.3.2 on page 48). To refer to the costs of an operation applied to a vertex v we use $c_d(v)$ for deletion, $c_i(v)$ for insertion, and $c_r(v, w)$ for renaming v to w.

Tree Edit Distance The TED applies a unit cost model that is based on the minimal number of edit operations required to transform one tree into another:

$$\gamma^{\mathrm{TED}}(v, v') = c \text{ (delete, insert, rename)}$$

Usually, the cost c for each type of edit operation is chosen to be a constant [41]. This cost model is only dependent on the number of operations on individual vertices.

Recent studies [19, 150] compare TED with alternative measures, noting that it matches trivial human intuition of differences. That is, the number of changes directly translates to the distance. It does not take into account the structural relations of vertices in their respective trees.

Fanout-Weighted Tree Edit Distance However, we often have to assume that the structure of a tree itself carries further information. Simply put, a vertex is not only defined by its own attributes, but also its parent. Therefore, whenever one vertex of a tree is edited, this implies a change of its descendants. While TED typically gives no weight to structural changes, the Fanout-weighted Tree Edit Distance (FTED) proposed in literature [21] uses the fanout of a vertex as an estimate for structural impact.

The FTED considers the fanout of each vertex to determine the cost of edit operations. The fanout of a vertex is the number of direct children. Hence, edit operations on a vertex

[1] Notably, this approach is *only* applicable in the context of our generator. It is not a general purpose tree distance for arbitrary trees.

with a large number of children have a high cost. This amplifies differences of vertices with many edges, and suppresses differences of vertices with few edges, particularly leaf vertices.

$$\gamma^{\mathrm{FTED}}(v, v') = \begin{cases} f_v + c & \text{if } v \neq \varnothing \wedge v' = \varnothing \text{ (delete)} \\ f_{v'} + c & \text{if } v = \varnothing \wedge v' \neq \varnothing \text{ (insert)} \\ \frac{f_v + f_{v'}}{2} + c & \text{if } v \neq \varnothing \wedge v' \neq \varnothing \wedge \lambda(v) \neq \lambda(v') \text{ (rename)} \end{cases}$$

Subtree-Weighted Tree Edit Distance The cost model of FTED is a simplification of structural impact to a finite context. This disregards recursive impact of changes on structure: Changing the definition of a vertex invalidates the definition of its children, which in turn invalidates their children as well. This can be severely underestimated by FTED, for example if a vertex has only one child that has a significant fanout.

Instead, we propose an exact cost model of recursive structural impact which we call Subtree-weighted Tree Edit Distance (STED). If a vertex changes, this affects all its descendants. Thus, the cost of operations is proportional to the size of the subtree of a vertex:

$$\gamma^{\mathrm{STED}}(v, v') = \begin{cases} |\,\mathrm{V}(\mathrm{T}(v)) \setminus \{v\}| + c & \text{if } v \neq \varnothing \wedge v' = \varnothing \text{ (delete)} \\ |\,\mathrm{V}(\mathrm{T}'(v')) \setminus \{v'\}| + c & \text{if } v = \varnothing \wedge v' \neq \varnothing \text{ (insert)} \end{cases}$$

Following ancestry considerations for tree-structured data, we further extend the cost of vertex mapping of STED for subtree move operations. For subtree move operations, we consider two use cases: local permutations as well as moves over long ranges.

Example 8.1. Let a tree describe a job according to the HEP computing model. In a job each task that is mapped to a vertex v within the tree, depends on its ancestry and most importantly on its parent $v.parent$. Dependencies between the different tasks are explicitly modelled by ancestors. Thus, the order of two tasks (including all their descendants) represented by their subtrees rooted at vertices u, v sharing the same parent can change without influencing all tasks or vertices that are descendants to u and v. However, moving a subtree rooted at vertex v from its parent to any another vertex $u \neq v.parent$, does indeed influence all subsequent tasks because their ancestry changes.

A local permutation does not change the ancestry of vertices. Thus, its weight depends only on the *width* of permutation, that is $\|u.pos - v.pos\|$, but not the subtree. In contrast, moving a vertex outside its local context changes ancestry, thus affecting the full subtree. Consequently, we extend the STED by the following cases and following refer to this cost model as Subtree-weighted Tree Edit Distance with Move (STEDM):

$$\gamma^{\mathrm{STED}}(v, v') = \begin{cases} \|v.pos - v'.pos\| c & \text{if } v.parent = v'.parent \text{ (permutation)} \\ \gamma^{\mathrm{STED}}(v, \varnothing) + \gamma^{\mathrm{STED}}(\varnothing, v') & \text{else} \end{cases}$$

8.1.2. Approximation Accuracy

Our approach to distance measurement is by design an approximation: The use of identities purposely introduces lossy compression. This enables sublinear complexity (see Section 4.5.3 on page 73), accumulation of attributes (see Section 5.3.1 on page 89) and is the motivation for ensembles (see Section 6.3.2 on page 107). Still, it means that our approach purposely ignores some information contained in trees.

However, the question is whether our approach preserves enough *relevant* information. To assess this, we evaluate how well our approach approximates exact edit distances. Notably, this is an evaluation only of the capabilities of our approach to represent structural differences – to the best of our knowledge, there is no established approach providing a comparable handling of attributes.

Distance Correlation

To estimate the applicability of our approach, we compare distance results to those of established cost models. However, since absolute distances are dependent on the algorithm implementing the specific cost models, a relative comparison is required. Our measure of choice is the correlation of distances for pairs of trees.

As we expect that our approach is a suitable replacement for edit distances, we use the linear correlation coefficient to relate distance measures. Broadly speaking, this is a measure whether two samples of values describing the same data have a linear relation. A correlation of 1 implies that the same relative information is present in both samples. On the other hand, a correlation of 0 implies that distinct information is represented by each sample. Ideally, a strong correlation between our tree distance and established ones shows that our approximation introduces negligible errors.

Evaluation of Correlation Our accuracy evaluation uses a sample of dynamic trees $(\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_n)$ from our recorded data. However, to stay consistent with edit based distances, we derive a set of static trees $\{T_1, T_2, \ldots, T_n\}$ that each include all vertices of their respective dynamic tree. This is the starting dataset for our distance correlation evaluation.

For each static tree T_j, we generate a number of variations $T_{j,1}, T_{j,2}, \ldots, T_{j,m}$ with well-defined changes (see Section 8.1.1 on page 130). In turn, for each distance measure k under consideration, we produce a vector $(\mathrm{dist}_k(T_j, T_{j,1}), \mathrm{dist}_k(T_j, T_{j,2}), \ldots, \mathrm{dist}_k(T_j, T_{j,m}))$, that is the distance from the initial tree to each variation. Notably, we do not derive distances between variations, as this would be unfeasible to calculate given the complexity of edit based distances.

For each distance k, we aggregate the distance vectors of all trees $T_1, \ldots, T_n$. This gives us a single vector $X_k = (\mathrm{dist}_k(T_1, T_{1,1}), \mathrm{dist}_k(T_1, T_{1,2}), \ldots, \mathrm{dist}_k(T_n, T_{n,m-1}), \mathrm{dist}_k(T_n, T_{n,m}))$ of size $n \cdot m$ for each distance. Thus, for two distances k and k', we derive the correlation as

$$\mathrm{cor}(k, k') = \frac{\sum_i \left(X_{k,i} - \bar{X}_k\right)\left(X_{k',i} - \bar{X}_{k'}\right)}{\sqrt{\sum_i \left(X_{k,i} - \bar{X}_k\right)^2}\sqrt{\sum_i \left(X_{k',i} - \bar{X}_{k'}\right)^2}}. \tag{8.1}$$

To estimate the expressiveness of our algorithmic approach and the various different identity classes, we following evaluate the distance correlation with respect to the established cost models as well as our introduced cost model STED and the supplementary cost model Subtree-weighted Tree Edit Distance with Moves (STEDWM) that builds on STED but supports the move of subtrees without any further costs.

Table 8.2.: Correlation of distance results for differing identity classes and distance functions. The correlation is calculated for different cost models: the unit cost model of Tree Edit Distance, the Fanout-weighted Tree Edit Distance cost model, our proposed Subtree-weighted Tree Edit Distance cost model, and the supplementary cost model Subtree-weighted Tree Edit Distance with Moves. Best correlation results for the specific cost models are highlighted. The correlation is analysed for three different distortion models including the insertion and deletion of vertices, the move of subtrees as well as a combination from all possible operations. Notably, Id^{P} is not evaluated for cost models STED and STEDM as it provides a direct implementation of these models. As such, the correlation for move distortions cannot be evaluated for Id^{P} as the distance is 0 in each case. In general, move distortions for STEDM result in 0 distance. Therefore, correlations for all identity classes are disregarded for move distortions.

			Distance Correlation			
Distortion	Distance	identity class	TED	FTED	STED	STEDM
Insert and delete	$\langle T\|1\|T'\rangle$	Id^{P}	0.55	0.53	0.69	0.69
		$\mathrm{Id}^{\mathrm{Pq}}$	0.62	0.60	0.73	0.73
		$\mathrm{Id}^{\mathrm{PqOrder}}$	0.62	0.59	0.72	0.72
		$\mathrm{Id}^{\mathrm{pq}}$	0.75	0.70	0.73	0.73
		$\mathrm{Id}_2^{\mathrm{pq}}$	0.72	0.68	0.73	0.73
		$\mathrm{Id}^{\mathrm{e,parent}}$	0.62	0.58	0.71	0.71
		$\mathrm{Id}^{\mathrm{e,noise}}$	0.63	0.60	0.73	0.73
	$\langle T\|\{M,D\}\|T'\rangle$	Id^{P}	0.79	0.82	1.00	1.00
		$\mathrm{Id}^{\mathrm{Pq}}$	0.85	0.87	0.98	0.98
		$\mathrm{Id}^{\mathrm{PqOrder}}$	0.84	0.87	0.98	0.98
		$\mathrm{Id}^{\mathrm{pq}}$	0.95	0.98	0.86	0.86
		$\mathrm{Id}_2^{\mathrm{pq}}$	0.84	0.97	0.86	0.86
		$\mathrm{Id}^{\mathrm{e,parent}}$	0.81	0.91	0.97	0.97
		$\mathrm{Id}^{\mathrm{e,noise}}$	0.84	0.87	0.98	0.98
Move	$\langle T\|1\|T'\rangle$	Id^{P}	–	–	–	–
		$\mathrm{Id}^{\mathrm{Pq}}$	0.73	0.66	0.73	–
		$\mathrm{Id}^{\mathrm{PqOrder}}$	0.68	0.62	0.68	–
		$\mathrm{Id}^{\mathrm{pq}}$	0.71	0.64	0.71	–
		$\mathrm{Id}_2^{\mathrm{pq}}$	0.72	0.65	0.72	–
		$\mathrm{Id}^{\mathrm{e,parent}}$	0.14	0.23	0.14	–
		$\mathrm{Id}^{\mathrm{e,noise}}$	0.71	0.63	0.71	–
	$\langle T\|\{M,D\}\|T'\rangle$	Id^{P}	–	–	–	–
		$\mathrm{Id}^{\mathrm{Pq}}$	0.96	0.88	0.96	–
		$\mathrm{Id}^{\mathrm{PqOrder}}$	0.82	0.80	0.82	–
		$\mathrm{Id}^{\mathrm{pq}}$	0.96	0.88	0.96	–
		$\mathrm{Id}_2^{\mathrm{pq}}$	0.96	0.88	0.96	–
		$\mathrm{Id}^{\mathrm{e,parent}}$	–	–	–	–
		$\mathrm{Id}^{\mathrm{e,noise}}$	0.95	0.88	0.95	–

Table 8.2.: Correlation of distance results (continued)

Distortion	Distance	identity class	Distance Correlation TED	FTED	STED	STEDM
Insert, delete, and move	$\langle T\|1\|T'\rangle$	Id^{P}	0.44	0.40	0.66	0.66
		$\mathrm{Id}^{\mathrm{Pq}}$	0.57	0.51	0.70	0.69
		$\mathrm{Id}^{\mathrm{PqOrder}}$	0.51	0.47	0.68	0.68
		$\mathrm{Id}^{\mathrm{pq}}$	0.76	0.66	0.60	0.57
		$\mathrm{Id}_2^{\mathrm{pq}}$	0.73	0.65	0.65	0.63
		$\mathrm{Id}^{\mathrm{e,parent}}$	0.50	0.46	0.67	0.67
		$\mathrm{Id}^{\mathrm{e,noise}}$	0.57	0.51	0.70	0.69
	$\langle T\|\{M,D\}\|T'\rangle$	Id^{P}	0.62	0.67	0.99	1.00
		$\mathrm{Id}^{\mathrm{Pq}}$	0.70	0.71	0.98	0.97
		$\mathrm{Id}^{\mathrm{PqOrder}}$	0.64	0.69	0.98	0.98
		$\mathrm{Id}^{\mathrm{pq}}$	0.94	0.93	0.75	0.70
		$\mathrm{Id}_2^{\mathrm{pq}}$	0.81	0.87	0.77	0.75
		$\mathrm{Id}^{\mathrm{e,parent}}$	0.67	0.77	0.96	0.97
		$\mathrm{Id}^{\mathrm{e,noise}}$	0.67	0.70	0.98	0.98

Comparison with Established Cost Models The correlation of our distances with various cost models for edit distances is shown in Table 8.2 on the previous page. We have used several distortion scenarios as well as several of our identity profile projection operators and identity classes. As expected, our naive identity profile projection operator 1 using only the presence of vertices is inferior to our identity profile projection operator taking into account both multiplicity M and duration D.

The best match for TED is our re-implementation of pq-grams, $\mathrm{Id}^{\mathrm{pq}}$. This is to be expected, as pq-grams are designed for this purpose. Notably, TED ignores most short and long range relations between vertices, which our specialised identities take into account. Our identity class using infinite-length encoding of parents and finite-length encoding of siblings $\mathrm{Id}^{\mathrm{Pq}}$ performs equally well as pq-grams if vertices are only moved. As we tested moves to replicate the behaviour of micro changes and out-of-order arrivals of events in streams, this is not surprising. Instead, this is to be expected as a simple permutation does not influence the structure of the trees.

Our distance measures based on our own identity classes generally correlate well with STED. This is to be expected, as STED implements the cost model we have based our identity classes on. In specific, the identity class using only the infinite-length parent encoding Id^{P} is a perfect match for STED under insertions and deletions. In contrast, the fixed-length parent encoding of pq-grams is inferior to express the impact of changes on subtrees.

There are three identity classes that we expect to be most suitable for our use case: The $\mathrm{Id}^{\mathrm{Pq}}$ is a strict consequence from our model of workflows, representing the definition of processes by their parents and sequences by their siblings. Consequently, this identity class correlates well with the strictly defined edit distances. In contrast, the $\mathrm{Id}^{\mathrm{PqOrder}}$ reflects the unpredictability of sibling sequences. This makes it naturally less correlated with strict edit distances, but performs well with distances allowing for permutations. Finally, the

ensemble identity class $\mathrm{Id}^{\mathrm{e,noise}}$ is a combination of the former two: It offers the precision of $\mathrm{Id}^{\mathrm{Pq}}$, being highly correlated with strict edit distances. At the same time, it incorporates the robustness of $\mathrm{Id}^{\mathrm{PqOrder}}$, performing well even for a multitude of types of differences.

8.1.3. Scalability

For scalability analysis we benchmark key performance characteristics of our algorithm. As our approach consists of the identity generation as well as distance calculation we analyse both components independently. This analysis mainly emphasises on runtime performance, as space complexity has already been shown in Section 4.5.3 on page 73. The runtime analysis of identity generation and distance calculation is performed with respect to available tuning parameters.

All benchmark evaluation have been performed using pypy version 2.0.2, which implements Python version 2.7. The operating system is Scientific Linux 6.7, the current standard used in HEP. Our analysis environment provides an Intel(R) Xeon(R) CPU E5–2640 v2, 2.00 GHz with 16 cores. In addition, the system provides 128 GB DDR3 RAM with a configured clock speed of 1600 MHz. The benchmark is not parallelised, meaning that a single core is used per run to determine key performance measures for identity generation and distance calculation.

Benchmarking

Our benchmarking of different identity classes is based on a random sample of different batch jobs of selected sizes of $50 \leq n \leq 250,000$ vertices from our dataset. For each distance calculation, the number of vertices of the two trees are approximately the same. We have evaluated our approach with respect to both identity classes and identity profile projection operators.

Figure 8.1 on the next page shows the runtime for generation and lookup of identities for different identity classes. In general, the accumulated runtime for each identity class is linear to the number of vertices, meaning that each identity is processed in constant time. This is to be expected based on our theoretical analysis of time complexity (see Chapter 4 on page 39 for an analysis of identity classes and distances as well as Chapter 6 on page 101 for ensemble). Thus, each of our identity classes is suitable for stream processing.

Our infinite-length parent encoding identity class $\mathrm{Id}^{\mathrm{Pq}}$ performs faster than the dynamic pq-grams, even for small extends in $\mathcal{P}$. This matches our expectation, as the recursive calculation of infinite-length encoding requires only information from the parent vertex. In turn, this allows our more complex identity class $\mathrm{Id}^{\mathrm{PqOrder}}$ to be as fast as the simpler pq-grams.

Finally, our use of ensembles gives notable speed advantages. The performance of $\mathrm{Id}^{\mathrm{e,noise}}$ is superior to the sum of $\mathrm{Id}^{\mathrm{Pq}}$ and $\mathrm{Id}^{\mathrm{PqOrder}}$. By calculating both ensemble identities together instead of separately, we gain roughly 25 % performance.

Figures 8.2a to 8.2b on page 139 show the impact of the distance measurements for multiplicity $\langle \mathscr{T}|M|\mathscr{T}'\rangle$, as well as multiplicity and duration $\langle \mathscr{T}|\{M,D\}|\mathscr{T}'\rangle$. We exclude the existence $\langle \mathscr{T}|1|\mathscr{T}'\rangle$, as it does not reflect our desired cost models (see Section 8.1.2 on page 134) but requires the same number of operations as $\langle \mathscr{T}|M|\mathscr{T}'\rangle$.

In general, the runtime of distance measurements is directly proportional to the number of events that are processed. The performance for a single event is roughly constant, but the number of events depends on the identity profile projection operator. Consequently,

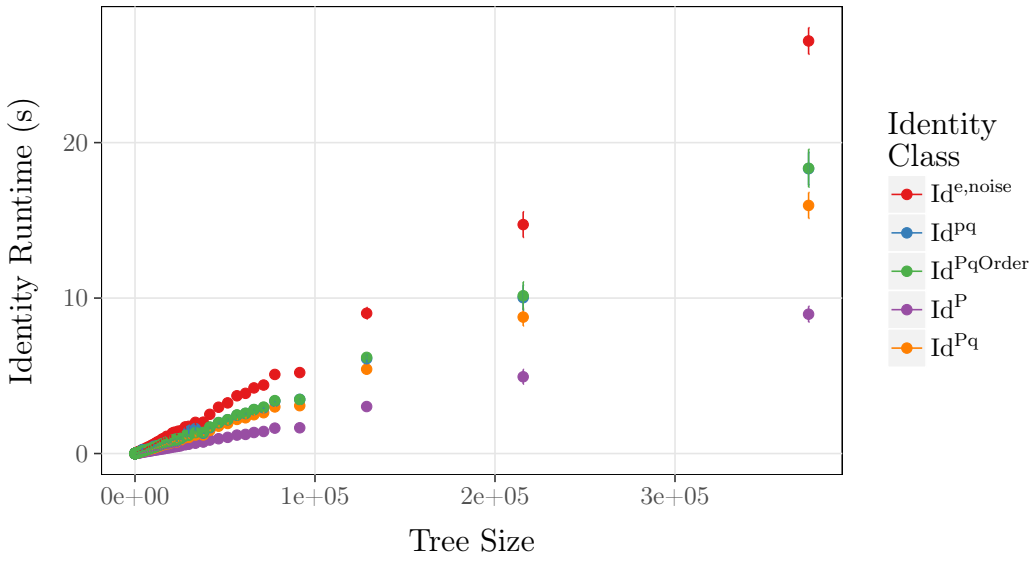


Figure 8.1.: Runtime behaviour of different identity classes for $\langle \mathscr{T}|M|\mathscr{T}'\rangle$. It can be seen that all identity classes scale linearly. The fastest identity class is Id^{P} while the slowest is the identity ensemble class $\mathrm{Id}^{\mathrm{e,noise}}$.

$\langle \mathscr{T}|\{M, D\}|\mathscr{T}'\rangle$ takes twice as much time as $\langle \mathscr{T}|M|\mathscr{T}'\rangle$. However, all of our distance measures are suitable for stream processing.

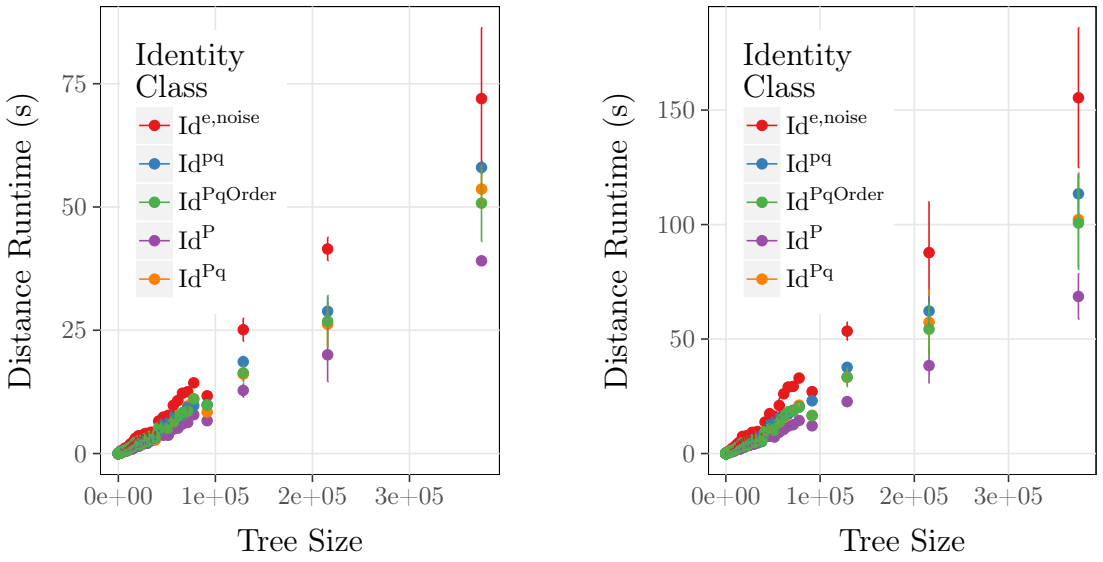
8.1.4. Sensitivity and Coverage

Our STED cost model uses the subtree size of a given vertex to reflect the intuition that vertices are defined by their ancestry. This implies that differences of inner vertices must be rated higher than differences of leaf vertices to derive an expressive distance. This meets the demand regarding sensitivity listed in Section 4.5 on page 67 with respect to the structure of a tree. To verify sensitivity of our proposed distance measure, we analyse relative distances with respect to a generated distortion.

Figure 8.3 on page 140 shows the difference in distance for changes on inner and leaf vertices, respectively. As desired and expected for our identity class $\mathrm{Id}^{\mathrm{e,noise}}$, changes on inner vertices are rated higher than for leaf vertices. This matches our definition of high-quality vertices with regard to process trees: inner vertices define all their descendants, and thus have a higher impact on distances.

In addition, the number of siblings included in identities has the inverse impact on inner and leaf vertices, rating the later higher. This is a result of the $\mathcal{P}$ dimension of identity classes not taking into account siblings of parents. Thus, increasing q to q' affects at most $q' - q$ siblings per change of vertex, but none of their children. For inner vertices, the number of included siblings is negligible compared to the size of subtrees. In contrast, changes to a leaf vertex only affect the leaf vertex itself and its siblings. Thus, the identity class $\mathrm{Id}^{\mathrm{e,noise}}$ allows to adjust the sensitivity for inner versus leaf vertices.

Our multi-step approach natively provides modular coverage. As demonstrated so far, the multiplicity distance covers different structural features of trees. Figure 8.4 on page 141 shows how coverage is easily extended by adding a distance operator sensitive to specific features. By construction, the multiplicity distance is indifferent to attributes. However,



(a) Runtime of $\langle \mathscr{T}|M|\mathscr{T}'\rangle$.

(b) Runtime of $\langle \mathscr{T}|\{M, D\}|\mathscr{T}'\rangle$

Figure 8.2.: Comparison of runtime for distance measures $\langle \mathscr{T}|M|\mathscr{T}'\rangle$ and $\langle \mathscr{T}|\{M, D\}|\mathscr{T}'\rangle$. While both distance measures show linear scalability, the identity profile projection based on $\langle \mathscr{T}|\{M, D\}|\mathscr{T}'\rangle$ does require more time as additional operations are performed for evaluating the duration of vertices.

our approach allows to simply add a new distance component that ensures coverage of desired features.

8.2. Applicability to High Energy Physics Jobs

The motivation of our work is the clustering and classification of HEP jobs, ideally at runtime. This is reflected by our choice of preferred cost model, identity class, and identity profile projection operator (see Section 8.1.2 on page 134). Thus, the evaluation of our approach with regards to HEP jobs is of key importance.

However, part of the reason for our work is the current lack of classification mechanisms for dynamic trees and in specific, attributed dynamic trees. Thus, there is no ground truth for us to compare our approach to. Still, we can manually derive a general classification for jobs of individual VOs based on the external monitoring provided by the Experiment Dashboard (see Section 2.2.2 on page 17).

8.2.1. Mapping of Experiment Dashboard Data

From monitoring at the different worker nodes at GridKa itself, no ground truth is available for the underlying workflows of batch jobs and their domain-specific results. Thus, evaluation of clustering as well as anomaly detection cannot rely on external quality criteria. However, there is no precise internal quality measure for density-based clustering producing arbitrarily-shaped clusters [45, 101]. Some measures seem promising [101] but score differently for varying use cases. Thus, having an external criteria to evaluate results especially for qualitative measures is crucial.

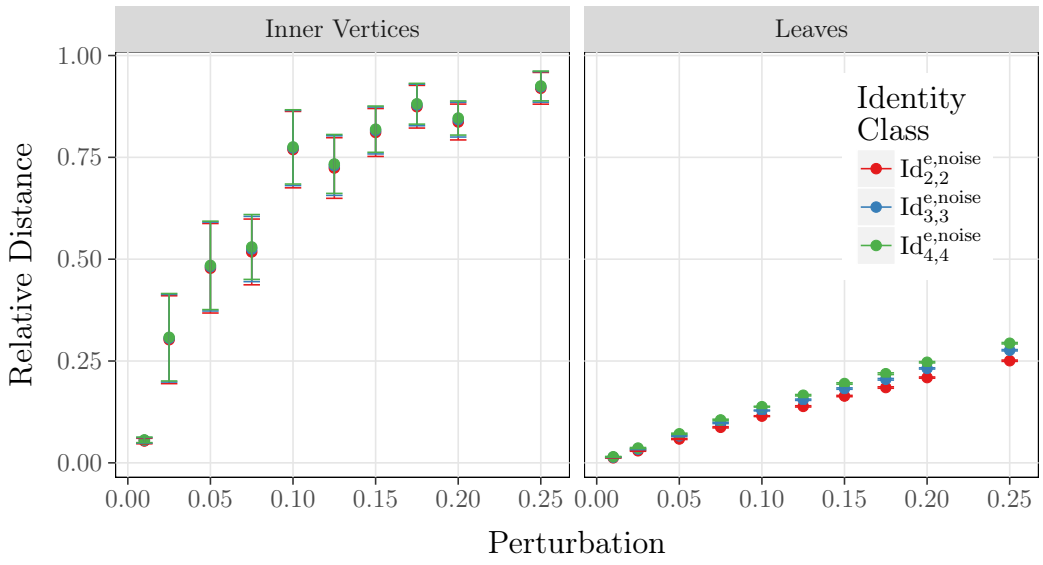


Figure 8.3.: Relative distance results of $\langle \mathscr{T}|M|\mathscr{T}'\rangle$ for insertion and deletion of vertices. The left plot shows the relative distance with regard to edit operations on internal vertices only. The right plot visualises the effect of edit operations on leaf vertices only. The sensitivity to edit operations is shown for the noise ensemble identity class with differing width, that is $q = \{2, 3, 4\}$. It becomes apparent, that edit operations on vertices that define the ancestry for other vertices have a higher influence on distance results. Thus, the distance $\langle \mathscr{T}|M|\mathscr{T}'\rangle$ is more sensitive to high-quality vertices.

The Experiment Dashboard at CERN offers an API to collect status information from listed payloads. Especially the application of job processing monitoring, namely the Task monitor, is of interest to supplement existing monitoring data with qualitative information about its underlying workflow and results. Thus, mapping this information to our recorded data has the potential to supplement monitoring data with qualitative results. For example, the state of a payload, reasons for failure of payloads, related workflows, or associated processing or simulation campaigns of the collaboration can be supplemented based on data from the Task monitor.

To realise a mapping between monitored data and information from the Experiment Dashboard all information of eligible jobs are queried. In the following, we refer to this dataset as *Dashboard data* while the dataset we collected within the GridKa is referred to as *monitoring data*. As Dashboard data is only available for the CMS experiment, we limit both datasets to one single experiment, namely the CMS experiment.

The mapping itself consists of two consecutive tasks. The first tasks focuses on estimating the difference in timing between the two datasets. Finally, we optimise the overlap of Dashboard data versus monitoring data by considering the estimated time shifts to find the best one-to-one matching.

Measuring Time Variations

Each payload from the Dashboard data is associated with a timestamp stating its start and end. We cannot expect this timestamp to match the timestamps from monitoring data

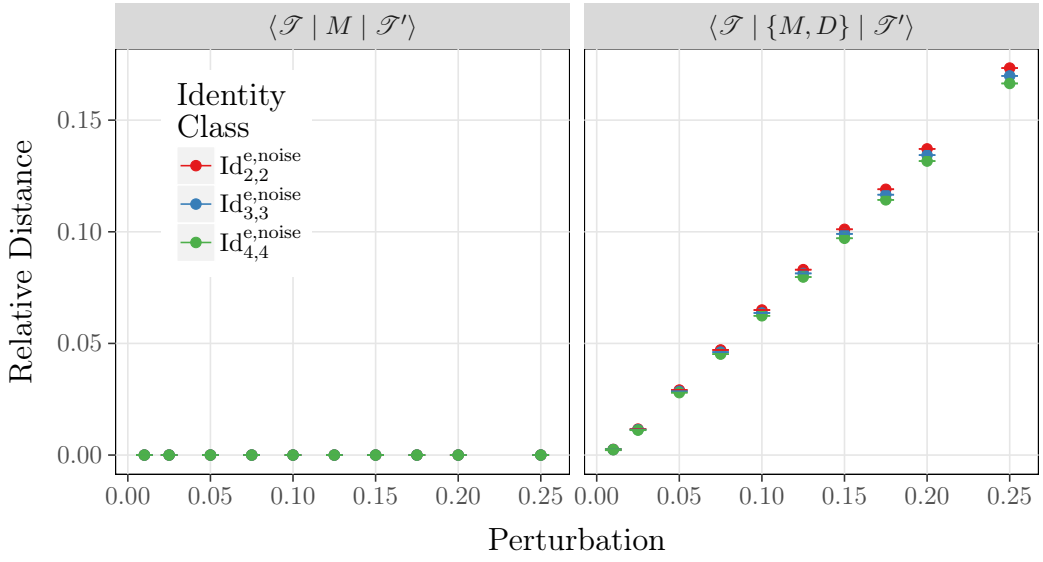


Figure 8.4.: Visualisation of influence of relative distance results of removal of attribute values regarding the distances $\langle \mathscr{T}|M|\mathscr{T}'\rangle$ and $\langle \mathscr{T}|\{M,D\}|\mathscr{T}'\rangle$. The left plot shows the distance with regard to a distance operator that ignores attributes. In contrast, the right plot shows the distance with regard to a modular addition of an attribute distance operator.

for several reasons: a) we have no knowledge on the time zone that is used to store the job status updates, b) the scope of reports is not known to us, thus we cannot expect the duration of the process encapsulating the payload in monitoring data to match the duration of the payload in Dashboard data, and c) we can expect differing latencies for different reports. Thus, we first measure the generic offset of timestamps and in addition, analyse variances of timestamps to limit the number of possible candidates for matching.

Payload Matching

Based on pre-calculated values for time offset and its variances, we limit the set of possible matches per payload. All matches are in a time range defined by our own timing information from monitoring data of jobs, and the average offset of timing information. With this limited matching possibilities, we can perform a naive search of best matches.

To perform the matching, for each payload from the monitoring data the matching set of payloads from Dashboard data is determined. We perform a recursive matching: After selecting the best matching Dashboard data for a monitored payload, we check for the selected Dashboard data whether there is a better payload to match. Once an ideal match is determined, we unwind the recursions, successively assigning the remaining optimal matches. This is repeated until all possible matches for monitoring data are found.

Characteristics of Mapping

Matching monitoring data and Dashboard data shows several features of our data: First, the monitoring data includes more payloads than the Dashboard data. This has several reasons: User workflows not utilising official frameworks do not necessarily follow the procedure

to report about job status. Also, defective payloads that do not start correctly may fail to submit a job status report. Second, several payloads do not correctly terminate their reports to the Experiments Dashboard. This triggers a timeout for the report, resulting in an exit timestamp corresponding to the maximum runtime of jobs. Both features show that the Experiment Dashboard is not suitable to serve as a job monitoring for sites. While it can supplement information derived locally, a dedicated monitoring by sites is required to have all relevant information.

Table 8.3 on the next page shows an excerpt of information available after matching monitoring data and available Dashboard data. The mapping adds supplementary information that cannot be derived from monitoring alone. However, this mapping does not provide a perfect ground truth for job characteristics. Information contained in Dashboard data does not cover all jobs, and there is no strict correspondence guaranteed between the two. Thus, we expect our monitoring data to contain features not described by Dashboard data, and classification not to perfectly replicate the Experiment Dashboard mapping.

8.2.2. Optimisation of Clustering

By mapping Dashboard data we enable the comparison of clusters to known external classes, promoting them to external quality measures. External quality measures evaluate the quality or accuracy of clustering algorithms by comparing the calculated clusters to a known ground truth [101]. External measures are more reliable and usually preferred over internal measures due to added objectivity. One of the most widely used external measures for clustering and information retrieval is the F-measure [159].

The F-measure is a single measure that merges *precision* and *recall* as their weighted harmonic mean. The precision and recall for a given class i and a calculated cluster j are given by

$$\text{precision}(i,j) = \frac{n_{ij}}{n_j} \tag{8.2}$$

$$\text{recall}(i,j) = \frac{n_{ij}}{n_i}, \tag{8.3}$$

where n_{ij} denotes the number of objects with class label i in cluster j, n_j denotes the number of objects in cluster j, and n_i denotes the number of objects in class i. The *balanced F-measure* equally weighting precision and recall is given by

$$F(i,j) = \frac{2 \cdot \text{precision}(i,j) \cdot \text{recall}(i,j)}{\text{precision}(i,j) + \text{recall}(i,j)}. \tag{8.4}$$

However, the balanced F-measure determines a score only for binary classification problems.

We, therefore, consider the *total weighted F-measure* to enable quality measurement for multi-class classification problems. The total weighted F-measure is the weighted average of all values for the maximum F-measure for each class, that is

$$\overline{F} = \frac{1}{n} \sum_i n_i \max(\{F(i,j) \mid j\}). \tag{8.5}$$

The values of F-measure lie in the range between $[0,1]$, and bigger values are an indicator for better clustering quality.

The $\overline{F}$-measure allows us to assess the quality of any existing clustering of our data. This enables the scanning of the parameter space for our clustering: Tables 8.5 on page 145

Table 8.3.: Supplemented monitoring data for payloads after matching Experiment Dashboard data. Several examples of matching monitoring data collected in GridKa to Dashboard data collected by CMS are shown. Matching is performed only using start and exit timestamps. Dashboard data adds for example a general category, the status of any payload that reported completion, and a description of the payload in a custom format.

Monitoring Data		Dashboard Data			
ID	Start / Exit	Start / Exit	Activity	Status	Campaign
1	1405032681	2014-07-10T22:51:24	reprocessing	SUCCEEDED	wmagent_pdmvserv_HIG-Fall13dr-00174_T1_ES_
	1405041685	2014-07-11T01:21:24			PIC_MSS_00261_v0__140623$_2$03651_1296
2	1405320849	2014-07-14T06:54:12	hctest	DONE	sciaba_output_c81gk9
	1405325225	2014-07-14T08:07:03			
3	1405443508	2014-07-15T16:58:39	production	SUCCEEDED	wmagent_pdmvserv_B2G-Summer12-00731_00152_
	1405453008	2014-07-15T20:20:00			v0__140713_161651_8002
4	1406845824	2014-07-31T22:30:31	analysis	unknown	peruzzi_nuovomateriale_V29A_splitmore_g25w7e
	1406847758	2014-07-31T23:02:35			
5	1406849464	2014-07-31T23:31:08	reprocessing	FAILED	wmagent_pdmvserv_SMP-Summer12DR53X-00006_
	1406863960	2014-08-01T03:32:38			T1_DE_KIT_MSS_00234_v0__140527_202346_5031

Table 8.4.: Selected campaigns from High Energy Physics for clustering. To consider relevant sizes of classes we condensed several versions of each campaign into one class. Each class is subdivided with regard to its outcome. We differentiate *failed* and *succeeded* payloads. Notably, we have different distributions of outcomes. For some classes, failures are seldom whereas other classes show more failures than successes.

		State	
Campaign	Workflow Type	Failed	Succeeded
alahiff_HCA-Spring14dr	Reprocessing	276	1511
alahiff_JME-Upg2023SHCAL14DR	Reprocessing	418	286
pdmvserv_SMP-Summer12DR53X	Reprocessing	105	237
vlimant_EGM-Fall14DR73	Production	1	754

and 8.6 on the next page show the respective $\overline{F}$-measure for various combinations of clustering parameters η and ϵ. To perform the analysis we selected specific HEP campaigns from our monitoring data. We expect the campaigns to represent individual HEP workflows. The selection of campaigns considers the inclusion of the two main types of workflows: production and reprocessing. An overview of data distribution to the single campaigns is shown in Table 8.4.

Based on the outcome of analysis from Section 8.1.2 on page 134 we identified the dynamic distance measure based on the identity profile projection operator $\{M, D\}$ as the most appropriate for our use case. We further consider the identity profile projection operator $\{M, D, \Lambda\}$ relevant. In comparison to the first option, the identity profile projection operator $\{M, D, \Lambda\}$ also considers the distribution of attributes. The characteristics of the identity profile projection operator Λ have not been evaluated so far, as no comparable method in the field of tree distance measures exists in literature. We, therefore, compare the two different identity profile projection operators based on the calculated $\overline{F}$-measure and identify the parameters that yield the best clustering for both options.

The ideal clustering for only structural information tends towards small distances, below 10 %. This indicates that similar nodes are very similar, and distinct nodes not similar at all. While this separates clusters well, it means that there is little room for distinctions inside clusters.

In contrast, clustering of trees with attributes yields a higher cutoff of ϵ. This suggests that clusters can be divided internally, allowing for sub-clusters to be detected. Additionally, the higher $\overline{F}$-measure indicates that separability is even higher when attributes are used as well.

Thus, the inclusion of traffic attributes is critical for two reasons: First, it improves our approach itself. We can derive a better separability, and clusters have better precision and recall. Second, it allows to reason about traffic usage by jobs. This allows us to classify workflows for different use cases, without prior knowledge.

Table 8.5.: Overview of calculated total weighted F-measures $\overline{F}$ for density-based clustering based on dynamic tree distance $\langle \mathscr{T}_1 | \{D, M\} | \mathscr{T}_2 \rangle$. To calculate $\overline{F}$, we consider different combinations of parameters η and ϵ. Best clustering results with regard to the determined activities based on Dashboard data are derived with $\eta = 2$ and $\epsilon = 0.1$ when considering identity profile projection based on identity profile projection operators D and M, that is the lifetime of processes as well as their multiplicity.

	η								
ϵ	2	3	4	5	6	7	8	9	10
0.05	0.457	0.456	0.452	0.304	0.304	0.304	0.307	0.307	0.306
0.1	0.678	0.677	0.674	0.674	0.361	0.361	0.361	0.361	0.361
0.2	0.290	0.290	0.290	0.290	0.290	0.290	0.290	0.290	0.290
0.3	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012
0.4	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012
0.5	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012

Table 8.6.: Overview of calculated total weighted F-measures $\overline{F}$ for density-based clustering based on dynamic tree distance $\langle \mathscr{T}_1 | \{D, M, \Lambda\} | \mathscr{T}_2 \rangle$. To calculate $\overline{F}$, we consider different combinations of parameters η and ϵ. Best clustering results with regard to the determined activities based on Dashboard data are derived with $\eta = 6$ and $\epsilon = 0.2$ when considering identity profile projection based on identity profile projection operators D, M, and Λ, that is the lifetime of processes, their multiplicity, and the distribution of attributes for network traffic.

	η								
ϵ	2	3	4	5	6	7	8	9	10
0.05	0.565	0.565	0.487	0.209	0.209	0.305	0.305	0.305	0.305
0.1	0.550	0.777	0.343	0.209	0.272	0.164	0.164	0.137	0.024
0.2	0.779	0.777	0.781	0.781	0.801	0.798	0.784	0.710	0.714
0.3	0.163	0.162	0.162	0.153	0.153	0.153	0.153	0.153	0.153
0.4	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.000	0.000
0.5	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012

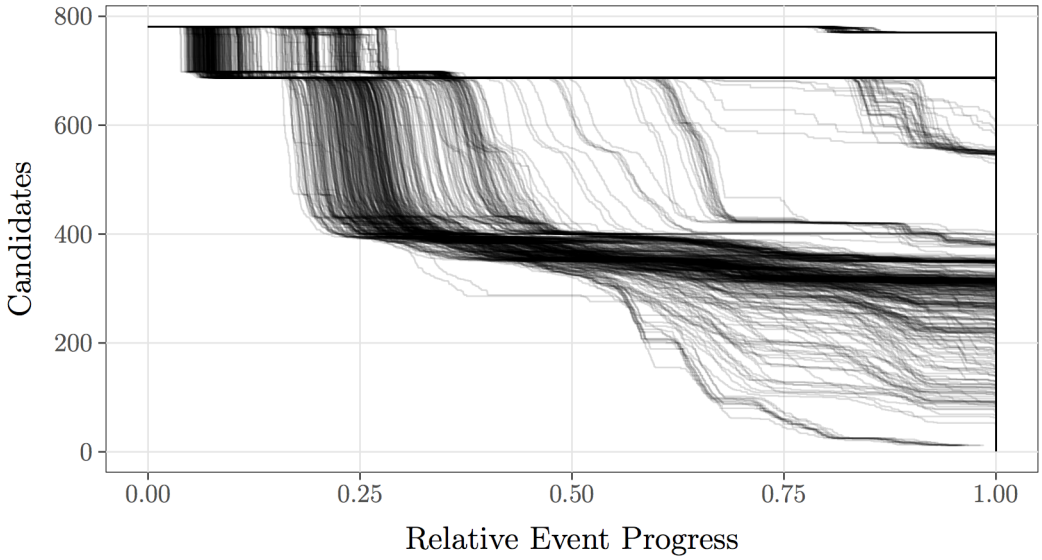


Figure 8.5.: Convergence of classification versus event progress, for trees which are not classified as outliers. The core nodes of clusters are chosen as candidates for classifiers to have sufficient statistics given our data sample. As trees unfold, their distance diverges from unsuitable candidates. Once the distance to a candidate is above an ϵ threshold used for neighbourhoods in clustering, the candidate is excluded from further comparisons.

8.2.3. Convergence of Classification

The classification of jobs consists of two stages: First, the identification of clusters in the known data, as shown previously. Second, the classification of monitored jobs as membership to a cluster. We express the later as the convergence and divergence of distances from the neighbourhood of each cluster (see Section 7.4 on page 121).

Notably, while we cannot guarantee early on that a job will converge to a specific cluster, we can detect early that it will diverge from others. This allows for some optimisations: Algorithmically, once a tree is guaranteed to be divergent from a cluster, we do not need to continue comparing tree and cluster. Logically, excluding a certain group of clusters allows us to negatively reason about a job – for example, that it is not a disk-intensive job.

Figure 8.5 shows the convergence of individual jobs to their respective core nodes from a given clustering[2]. The number of core nodes that remain as candidates drops to 50 % already after a relative event progress of 25 % for many jobs. As such, much less core nodes must be compared over the full lifetime of a job than there are actual nodes. This supports our assumption, that the proposed approach for online analyses of dynamic trees is advantageous for streaming environments as the number of required calculations can be efficiently reduced.

[2]We consider the convergence of jobs to core nodes in this evaluation as we require a sufficient amount of trees for statistical relevance. In a real scenario the CRs of clustering are considered. This requires even less calculations than anticipated in this example.

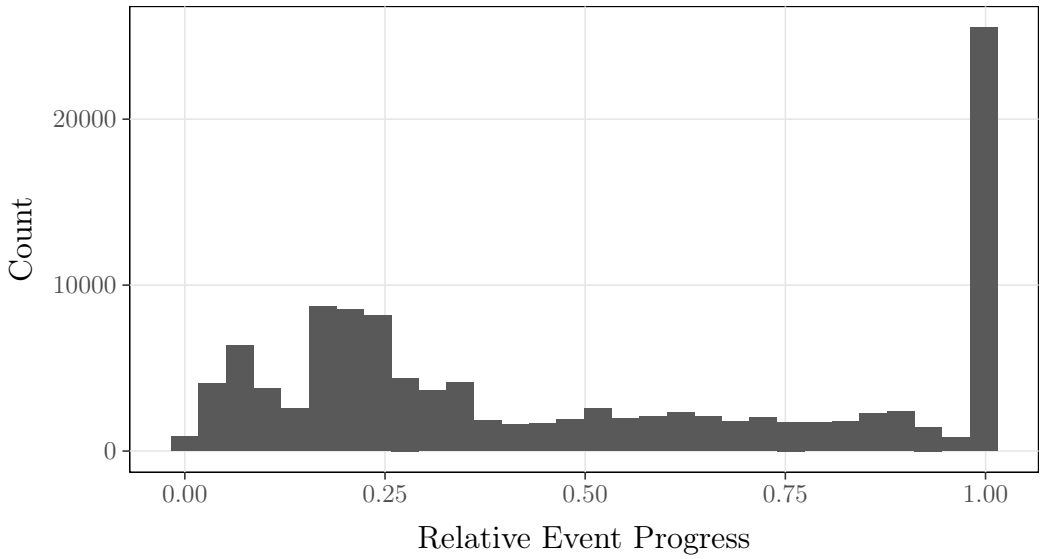


Figure 8.6.: Visualisation of distribution of detected anomalies for distance comparison between two diverging trees $\mathscr{T}$ and $\mathscr{T}'$. The distribution is shown with regard to the event progress. Approximately 54.30 % of tested trees are detected within 50.00 % of streamed events. The peak at the end of the streamed events is justified by the number of events of the observed tree $\mathscr{T}'$. Whenever the $\mathscr{T}'$ has much less events than the recorded tree $\mathscr{T}$, divergence either requires a sufficient amount of mismatching events or the completion of $\mathscr{T}'$ to safely determine missing vertices from $\mathscr{T}$ while retaining monotonicity.

8.2.4. Detecting Anomalies

Early detection of anomalies is an important use case for monitoring jobs. On the one hand, it allows for targeted countermeasures without affecting all batch jobs of a VO. On the other hand, detecting misbehaving payloads while they are running enables live debugging.

Conceptually, we define an anomaly as any job that does not match established behaviour. In other words, the respective dynamic attributed tree diverges from all known clusters. Due to the monotonicity of our divergent dynamic distance, this means we can identify anomalies early.

Figure 8.6 shows the fraction of a tree we must track to be sure it is anomalous. Notably, of the anomalies not identified during runtime, most are smaller trees than the CR they are compared against. Thus, while they are detected late according to their own time, they are in fact detected early in terms of the recorded tree sample. For this sample, 54.30 % of jobs are detected as outliers before less than 50 % of their events have been processed.

The approach considered so far is completely unsupervised and does not require any intervention of human resources. However, we can significantly improve the outcome of our anomaly detection by considering supplemented monitoring data to recognise failed payloads as anomalies. This is advantageous for users as well as operators as more jobs can be acted on early.

In Table 8.4 on page 144 it can be seen that a substantial amount of payloads can be identified as anomalies based on this extended definition of anomalies for HEP. This shows how our approach enables the combination of autonomous monitoring and external

information to derive superior classification with a semi-supervised learning approach.

8.3. Summary

In this Chapter, we have evaluated our approach for key characteristics and its quality of outcome with regard to a real use case. The former part is dedicated to the formal performance of our approach. The later part shows the applicability of our approach to realistic scenarios.

As our approach is by design an approximation, we have evaluated its conformity with exact edit distances. When compared against edit distances taking subtrees into account, our distances are close to an optimal match. Most importantly, an ensemble using both an exact and unordered view of siblings has proven to be an accurate and robust approximation.

We have evaluated the runtime characteristics of both identity classes and identity profile projection operators. This shows that all our approaches are constant per event, thus making them suitable for stream processing. Additionally, both sensitivity and coverage are adequate for our use case.

Evaluating our approach for the use case of HEP jobs requires a reliable classification for comparison. As no exact comparison is available, we have collected monitoring data from the CMS Experiment Dashboard. This allows us to create a mapping of payloads to workflows.

We use the classification to the derive the $\overline{F}$-measure, which expresses both precision and recall. This allows for a scan of clustering parameters, and in turn the derivation of optimal parameters. The results show that attributes are important for separability.

Finally, we use a reference implementation of our online clustering to demonstrate characteristics of our approach. In specific, we evaluate the temporal evolution of our classification. On the one hand, this shows that only a fraction of theoretically possible comparisons needs to be processed. On the other hand, it demonstrate how we can detect and react on anomalous jobs during their runtime.

9. Conclusions & Outlook

Tree structures are a natural format to express hierarchical data. However, the complexity of such data makes its efficient and precise handling highly non-trivial. Especially for comparing trees, exact approaches have super-linear complexity, while approximate approaches make strong assumptions on structure, sequence, and types of data. However, realistic tree structures may be large, change over time, contain arbitrary data, and be provided in an arbitrary order. The work presented in this thesis is centred on an approach and formalism to efficiently measure and work with similarities and distances for attributed dynamic trees in streaming environments.

The motivation for this work is the monitoring of HEP batch jobs. Recently, this has been complicated by the adoption of the pilot paradigm. Pilots are placeholder batch jobs that acquire resources and fetch batch jobs for execution by themselves. This creates an overlay batch system over existing batch systems. The efficient inspection, evaluation, and finally classification of jobs contained in pilots brings up a number of challenges.

First, we have assessed means and requirements to identify jobs in this environment. The technical setup necessitates a monitoring of the process hierarchy of batch jobs to identify pilots and recognise contained jobs. Also, the major categories of HEP workflows, namely simulation and reconstruction workflows, suggest that network traffic is key for separating workflows. However, as we operate in a production system dedicated to the processing of workflows, any approach we consider must be limited in space and processing resources. Thus, we have devised a sensor creating a monitoring stream of process and network traffic data as streaming, dynamic trees.

Working with dynamic trees in streams in this environment requires an efficient approach, and a formalism reflecting it. As part of this thesis, we propose a new formalism to express established decomposition-based approaches to measuring tree distances. Our formalisation has two goals: First, a generalisation of decomposition-based distance methods. We have shown that we can easily replicate common distance measures from literature with our formalisation and provide our distance measures as well. Second, our formalisation naturally translates to dynamic trees and incremental distances. This makes all our distance measures suitable for stream processing of dynamic trees.

A distinct feature of our use case is traffic monitoring for processes, expressed as attributes of vertices in our dynamic trees. As such, we have focused on integrating distances based on attributes into our approach. In contrast to most existing approaches, we describe attributes as vertices with distinct, attached values. This is the basis for integrating attributes into our approach, accumulating dynamic attributes using statistics. This aggregation allows us to derive continuous distances from attribute values.

Relations in trees are necessarily complex, and the black box nature of our use case means that many relations are unknown. Motivated by this, we have introduced an ensemble method to combine individual, simple distance measures to express complex relations. Ensembles allow the independent but parallel evaluation of multiple distance measures. We have presented and used ensembles to suppress noise occurring in the stream of tree event data, without sacrificing precise distance evaluation.

The ultimate goal of our monitoring and comparison of dynamic tree data is a meaningful classification of tree data in near real-time to enable outlier detection of batch jobs. We provide a classification of dynamic trees by using an incremental clustering. Aside from choosing a suitable clustering mechanism, we propose three optimisations targeting the online analysis of dynamic trees. First, we extend the density-based clustering approach DenGraph for CRs to efficiently represent arbitrarily shaped clusters. This, in turn, enables the reduction of required distance calculations for incremental clustering. Second, we introduce the concept of virtual nodes to our clustering. The nodes of monitored jobs are inserted tentatively into our clustering mechanism, allowing them to be classified without skewing existing clusters. Third, we provide an approximated normalised incremental divergent distance for dynamic trees. Using constraints of the clustering and virtual nodes, we can reliably exclude clusters during the runtime of jobs.

Finally, we have evaluated multiple aspects of our approach. To avoid bias, instead of synthetic data we use a hybrid approach combining recorded data from monitoring together with well-defined test scenarios. Our benchmarking shows that our approach is linear in time and sublinear in space complexity concerning the length of streams. This means that we can, in principle, process tree-structured data of arbitrary size. Furthermore, we have conducted a correlation analysis to compare our distance approximations to different tree edit distance cost models. This correlation analysis shows that our approach is a viable replacement for exact tree distances.

Finally, we have demonstrated the interaction of all proposed components to facilitate an online analysis and outlier detection for our use case. By comparing our clustering with external quality measures, we have shown that our approach replicates the categories of workflows and even recognises sub-structures. Building on this, we have been able to demonstrate early outlier detection based on our divergence approach, as well as the superior sensitivity introduced by our handling of attributes in dynamic trees.

9.1. Future Applicability and Extensions

Our approach and formalisation for distance measurements on attributed dynamic trees in a streaming context provide a strong foundation for future research. With its modularity and extensibility, we can further investigate more complex decompositions, distances, ensembles, and optimisations.

So far, we have restricted ourselves to simple decompositions, reflecting the limited knowledge about our data. Given the workflow established in the course of this thesis, we can in the future exploratory approach extended decomposition strategies, reflecting all three decomposition dimensions we have identified for stream analysis. Instead of pre-defined decompositions, an automatic learning of relevant decompositions could provide valuable insights. Furthermore, such an automatism would allow applying our approach to other fields with little effort.

Our approach to handling attributes already provides an efficient, precise means of expressing features of dynamic trees that are missing from most tree distance approaches. Still, our focus on streaming capabilities means that we have forgone the adoption of more complex representations common in time series analysis. This offers the opportunity to either integrate complex time series analysis with hierarchical structure or to find more complex approximations suitable outside of streaming environments.

Future optimisations are required to support additional granularity of decomposition

and features. While we can show that sublinear or even constant complexity can be expected for non-degenerate trees, we cannot guarantee it. Introducing sketching methods to handle identity profiles would guarantee minimal space complexity even for use cases relying on large or unlimited identity alphabets. To express complex attributes, skewed distributions could balance the generality of MultisetStatistics and simplicity of incremental PDF statistics.

The efficiency of our approach allows us to perform online analysis even without leveraging the distributed environment to our advantage. Only the collection of data by our monitoring sensors is parallelised and distributed over the entire batch system. Our concept of virtual nodes already supports concurrency of classification, and could further be parallelised. Ideally, the classification of streamed trees and subsequent clustering would be distributed by parallelising the clustering itself. Such a distribution of clustering allows for a multi-agent approach to classifying jobs and detecting anomalous workflows.

Finally, our approach is not limited to the use case of HEP batch job monitoring. It is an efficient tool for comparing any hierarchical data, especially if it is dynamic or features attributes.

A. Software Tools and Frameworks

In the following, a brief description on the tools that have been implemented during the course of this thesis is given. A more specific documentation can be found online. Each of the tools is published under version control and includes a range of unit tests to demonstrate functionality and ensure validity.

A.1. ASSESS

ASSESS is the reference implementation of proposed distance measures and optimisations defined upon. This application provides a framework to test the different distance measurement approaches for dynamic trees as well as static trees. The application is based around the concept of events. Each event describes a single change within a tree. Therefore, we differentiate between three kinds of events:

- events to append a vertex,
- events to remove a vertex, and
- events to change an attribute of a given vertex.

Each event is handled independently. Depending on the current configuration, different actions are performed based on a given event. The main components can be divided into:

- algorithmic components, and
- analysis components.

The algorithmic components offer methods to analyse distances and similarities of dynamic as well as static trees. The analysis components offer possibilities to prepare the calculated data as well as its performance for further analysis.

The main class within algorithmic components is the `TreeDistanceAlgorithm`. It relies on two components:

- an identity class, and
- the distance functionality to be used.

Each of those components is implemented by different methods providing differing complexities and precision.

The identity class implements the identity building process. We implemented methods from literature as well as our own approaches. The most complex one considers the ensemble of identity classes that allows to combine different identity classes into one single identity class.

Another important component that is provided by the framework are different kinds of decorators. The most important part of the framework is to compare different approaches against each other. Therefore we provide different decorators that support the generation of different output that can further be analysed. The most important decorators include:

- *Distance decorators*: distance decorators take care on preparing the actual distance calculations for further usage. Those include the preprocessing for matrix-based comparison but also for incremental distances for each possible combination of identities.
- *Identity decorators*: identity decorators allow the logging of generated identities for further analysis.
- *Anomaly decorator*: the anomaly decorator prepares information about the information if an anomaly has been detected.
- *Performance decorators*: performance decorators allow to measure runtime of different functionalities like identity building process, distance calculation or the approach as a whole.
- *Compression decorator*: the compression decorator prepares key data for input sizes as well as compressed sizes based on identity class in use.

A.2. BPNetMon

The application BPNetMon is based on the ideas of NetHogs - a small 'net top' tool, grouping processes by bandwidth instead of breaking the traffic down per protocol or per subnet. Detailed information about NetHogs are available at SourceForge.

Preliminaries and Build The usage of GridKa Monitor requires the `libpcap` packages to be installed on the system. The application furthermore requires root access for execution. The application can be build by utilizing SCons. It comes packaged with the local version 2.3.0. It can be used if SCons is not installed locally.

To make a clean build the command `scons-local-2.3.0/scons.py -c` can be considered. The executable is located at `build/your_system/gridka_monitor`.

We already tested the application for different Linux flavours including for example:

- Debian,
- Scientific Linux, or
- Ubuntu.

Parameters

Loading the External Configuration File The concrete execution can be configured via a configuration file called config. It is expected to be stored at `/etc/sysconfig/gnm`. If you want to specify another folder you can do so with the parameter `l`. The configuration file can contain different named groups that might be loaded. To declare which group is used the parameter `c` is given when execution the application. The

parameter `c` expects the name of the group containing the different configuration parameters: `sudo gridka_monitor -l /your/location -c gridka`.

Setting the Pid for Grouping Processes The grouping pid can be set when starting the monitoring tool by using the parameter `g`: `sudo gridka_monitor -c eileen -g $(pgrep -n sge_execd)`.

Setting the Name for Grouping Processes To allow an easy installation and monitoring the grouping process can also be determined by its process name with parameter `n`. If a grouping name is specified the grouping pid has no influence. When this parameter is used the tool ensures to re-initialise the monitoring after a restart of grouping name: `sudo gridka_monitor -c gridka -n "sge_execd"`.

Setting the Log Interval The log interval can also be overwritten besides what is being specified in the configuration file. This is done by specifying the log interval with parameter `i` and some value in seconds > 0: `sudo gridka_monitor -c gridka -i 20`.

Configuration Options

groupingpid `pid_t`: The option `groupingpid` specifies the maximum valid pid being taken as ppid to group the single processes. For a better handling and dynamic adaptions the `groupingpid` can also be given as input parameter of the application itself. For example the parameter should be initialised with the pid of the shepherd process to group the processes by batch jobs being started by the shepherd process.

groupingname `string`: The option `groupingname` specifies the process that is being taken to group the single processes. The first process that is found is taken into account for grouping. Others are being ignored. As soon as the process is finished, the tool gets into waiting state and looks for new starting processes with process name `groupingname`.

skipotherpids `bool`: Next to the grouping of jobs there might also be background traffic by other processes in the process tree. By activating `skipotherpids` this traffic is not being monitored. By setting `skipotherpids` to false the traffic is going to be logged independently from the grouped processes.

loginterval `unsigned int`: The option `loginterval` specifies the intervals being logged to in seconds.

detail `processconnection | process | tree | treeconnection | group | job`, or `groupconnection`: The detail controls the amount and grouping of information being logged. The option `processconnection` logs all connections of every process not being skipped. By defining `tree` all processes are logged and additionally the traffic owned by a process is accumulated and logged. The option `treeconnection` logs details about the connections belonging to a process. By defining `process` only the accumulated traffic and its processes is logged. The option `group` accumulates the traffic of all connections and subprocesses of a grouped process. The last option `groupconnection` logs every connection belonging to the process and subprocesses being inside a single group.

The last option `job` is a special case implemented for sending information to UGE. It identifies the job ID and regularly sends aggregated job information to the batch system.

A.3. DenGraph

The implementation of DenGraph implements the density-based clustering approach described in the papers Schlitter, Falkowski, and Lässig [192, 193]. We further extend the implementation by the notion of CRs based on our representation of identity profiles. This enables the reduction of necessary distance calculations as our use case otherwise requires a fully-meshed representation.

A.4. Tree Generator

The Tree Generator takes a stream of vertices and attributes and enables the execution of changes based on those events. Each change including the insertion, deletion, renaming, or move of vertices is tracked within the generator. Based on a specified cost model, specific distances are calculated given the history of changes that were applied to each vertex as well as its ancestry.

To enable specific results of tree distortion a probability can be given that determines when a change is introduced. In a second step the change to be executed is evaluated. Therefore the Tree Generator can be configured with different probabilities for each possible change operation.

We further introduced another possibility to specify limit or specify on which part of the tree a given change should be executed. This is enabled by introducing a filtering mechanism to determine if a specific vertex is considered for analysis if it should be changed. This enables to implement a distortion based only on leaf vertices, inner vertices, or vertices containing attributes.

B. Configuration and Evaluation Workflows

B.1. Batch System Monitoring

```
1  [trees]
2  intern = ^10\.\d{1,3}\.\d{1,3}\.\d{1,3}$|192\.108\.4[5-9]\.\d{1,3}$
3  extern = .*
4  splitinternextern = true
5  skipotherpids = true
6  loginterval = 20
7  detail = treeconnection
8  logdir = /var/log/gnm
9
10 [uge]
11 intern = ^10\.\d{1,3}\.\d{1,3}\.\d{1,3}$|192\.108\.4[5-9]\.\d{1,3}$
12 extern = .*
13 skipotherpids = true
14 loginterval = 300
15 detail = job
16 groupingname = sge_execd
17 logdir = /var/log/gnm
```

Listing B.1: Configuration in use at GridKa. The configuration named *tree* is used for online analysis of batch jobs whereas the configuration named *uge* is used to exemplarily report monitored data to the Univa® Grid Engine® batch system. The most relevant difference between the two configurations is the detail. While the former configuration monitors data for each process within the Unix process tree the latter aggregates all data for one single process.

B.2. Workflows

The results and argumentation presented in this thesis are based on several workflows. Each of these workflows has a specific objective to produce results and support argumentation. In general, each workflow can be divided into three tasks:

- data selection and generation,
- data processing, and
- data analysis.

Each of these tasks can be composed of several steps. Furthermore, the workflow mechanism we utilise supports the reprocessing of specific steps of a given workflow. This is especially useful when certain criteria, for example in ASSESS, are changed to prevent from re-processing of unchanged steps.

The steps of the workflows are executed sequentially as each step depends on some input data and the input data is often given by the previous step. Whenever possible,

the processing of the steps itself is parallelised. We therefore consider different levels of parallelisation: distribution to several machines as well as multi-core processing.

Each of the workflows producing qualitative analysis results is based on one data preprocessing workflow that targets the processing of raw data we recorded by utilising BPNetMon to derive specific payloads or pilots.

In the following, the preprocessing workflow as well as the data analysis workflows along with their configuration are briefly presented.

Index by ... Usually the first step considered in workflows. This creates a key-value index by a specified criteria such as number of vertices, number of attributes or something else. The key following holds the number of vertices for example while the value usually stores information on the file path where the given tree can be read from.

Subset data The activity to subset data requires a specification of the criteria how to build the subset. This might contain a range of values the number of vertices is expected to fall into. Keys that do not match the specified criteria are not considered for further steps of the workflow.

Squish index The activity to squish an index takes care on grouping keys based on a dynamic binning approach. This dynamic binning can further be parameterised by specifying the maximum number of keys that may fall into the bin as well as the maximum distance between single keys. The new key becomes the mean value of all keys inserted into a group. All values are summarised into one list.

Select data Data selection works on the specified index. For each given key in the index the data selection is performed. For each key given values for *count* and *repetition* are evaluated to determine the final values that are considered in the following steps of the workflow.

Aggregate data To aggregate the whole index of key-value pairs into one single bin by skipping the available keys, this method can be used.

Perturbate tree The selective distortion of trees is key for many analyses to properly analyse characteristics. Without aimed distortions qualified reslts would be hard to derive based on real data only.

Process as matrix, Process as vector The methods to process distances as matrix or vector are part of the key functionality of the ASSESS reference implementation. The first method ensures that for each given tree the pairwise distance is calculated. Furthermore the calculations can be optimised by defining if distances are symmetric and therefore some calculations might be skipped. The second method calculates single distances for a given observed tree to a list of recorded trees.

Furthermore, many specific methods are available. For example the plotting of results, a specific analysis of results, or even the clustering based on pre-calculated distance measures.

Pre-processing

Before executing the different workflows the data from monitoring are prepared for frequent processing. This pre-processing includes the indexing of pilots and payloads to enable search for specific features such as related VO or time of execution.

We further analyse the data to exclude incomplete monitoring data. Incomplete data is recorded for example when starting or stopping the monitoring. As the batch job has already been started or is not finished yet, it cannot be guaranteed that all information are available.

The repeated analysis of pilots and payloads further makes it necessary to access individual data. We therefore extract single payloads of CMS pilots and save them for convenient processing.

Dataset Statistics

The workflow is about gathering statistical information describing the whole dataset of pilots as well as CMS payloads. We are interested in statistics regarding data without attributes, and data with attributes. This enables an improved comparison to methods for static trees not supporting time series of attributes values.

To summarise only valid data, we will filter the input depending on the number of vertices. Each tree with less than 100 vertices is being skipped from the final analysis.

Specifically, this workflow collects the following data by considering each for the two variants given attributes or no attributes:

- number of vertices,
- fanout, and
- depth of leaf vertices.

Furthermore the data on

- duration of tree,
- number of vertices with attributes, and
- number of events per attribute

are collected. After relevant data is collected, analysis regarding the distribution of relevant values is executed.

#	Input	Description	Configuration
1	–	Index by |vertices|	All worker nodes
2	–	Index by |vertices|	All payloads
3	1, 2	Subset data	$\|\mathscr{T} > 100\|$
4	1, 2	Calculate statistics	
5	4	Prepare plot	
6	4	Analyse distributions	

Diamond Analysis

Diamonds can occur in tree decomposition methods targeting fixed-length identity encoding. We therefore analyse available data for different effects of diamonds regarding the fixed-length encoding of dynamic pq-grams for different parameters.

The analysis itself targets the number of diamonds, the number of nested diamonds as well as the perturbation that is introduced.

#	Input	Description	Configuration
1	–	Index by |vertices|	
2	1	Squish index	
3	2	Subset data	$25 < \|\mathscr{T}\|$
4	3	Select data	Seed fixed Repetition: 10 Count: 1
5	4	Analyse diamonds	Identity class: $(\mathrm{Id}^{\mathrm{pq}}_{1,0}, \mathrm{Id}^{\mathrm{pq}}_{2,0})$, $(\mathrm{Id}^{\mathrm{pq}}_{2,0}, \mathrm{Id}^{\mathrm{pq}}_{3,0})$, $\ldots$, $(\mathrm{Id}^{\mathrm{pq}}_{7,0}, \mathrm{Id}^{\mathrm{pq}}_{8,0})$
6	4	Analyse diamond perturbation	*Compare step 5*
7	5, 6	Prepare plot	

Distance Progress

This workflow is intended to visualise the progress of distance calculation while incrementally checking the distance to different combinations of trees:

- $\mathrm{dist}(\mathscr{T}, \mathscr{T})$,
- $\mathrm{dist}(\mathscr{T}, \mathscr{T}')$, with $|S(\mathscr{T})| = |S(\mathscr{T}')|$, and
- $\mathrm{dist}(\mathscr{T}, \mathscr{T}')$, with $|S(\mathscr{T})| < |S(\mathscr{T}')|$.

For better comparison, the recorded tree is equal for each tested case. Therefore, the influence of incremental distance results can directly be compared.

#	Input	Description	Configuration
1	-	Index by |vertices|	worker node = c01-007-106
2	1	Subset data	$900 < \|\mathscr{T}\| < 1100$
3	2	Aggregate data	
4	3	Select data	Seed fixed Repetition: 1 Count: 1
5	4	Process as matrix	identity class: Id^{P} similarity: $\langle S^{\mathrm{simple}}(\mathscr{T})\|M\|S^{\mathrm{simple}}(\mathscr{T})\rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute distance vector
6	4	Perturbate tree	Seed fixed base probability: 10 % delete probability: 33 % insert probability: 33 % edit probability: 33 %
7	6	Process as vector	identity class: Id^{P} similarity: $\langle S^{\mathrm{simple}}(\mathscr{T})\|M\|S^{\mathrm{simple}}(\mathscr{T})\rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute distance vector
8	4	Perturbate tree	Seed fixed base probability: 30 % delete probability: 100 %
9	8	Process as vector	emphCompare step 7
10	5, 7, 9	Prepare plot	

Tree Generator Validation

To show the validity of our approach to tree generation by introducing distortions into a given tree, we introduce different distortions into selected trees. The selected trees are a subset of available data with a limited amount of vertices only to provide the possibility to calculate exact distances based on TED. For each chosen tree and its distortion exact distances for TED are calculated. The calculated distances measured by an external implementation as well as the derived distances are compared to validate the quality of derived measures.

Results show, that our tree generator correctly estimates an upper bound for exact tree edit distance.

#	Input	Description	Configuration
1	–	Index by |vertices|	
2	1	Squish index	
3	2	Subset data	$100 < \|\mathscr{T}\| < 500$
4	3	Select data	Seed fixed Repetition: 1 Count: 10
5	4	Perturbate tree	Seed fixed Base probability: 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 Insert probability: 33 % Delete probability: 33 % Edit probability: 33 %
6	5	Process as vector	Identity class: Distance: TED Analysis: absolute, normalised distance
7	4	Perturbate tree	Seed fixed Base probability: 0.05, 0.1 Move probability: 100 %
8	7	Process as vector	*Compare step 6*
9	6, 8	Calculate correlation	

Distance Correlation

Given the validity of our Tree Generator (compare Section B.2 on the preceding page) we create different distance based on several cost models to show the correlation of our proposed distances. We therefore select a number of trees. For each tree a set of distortions is introduced, each representing different available scenarios such as permutation of vertices. While distorting the given trees the number of edit operations are counted within the Tree Generator for further utilisation in the distance correlation estimate.

Each vector of correlated trees as well as the original tree the distortions are based on are analysed regarding different identity classes as well as distance measures. The resulting vectors of distance results are considered in the following for correlation analysis.

#	Input	Description	Configuration
1	–	Index by |vertices|	All payloads
2	1	Squish index	
3	2	Subset data	$\|\mathscr{T}\| < 15\,000$
4	3	Select data	Seed fixed Repetition: 1 Count: 5
5	4	Aggregate samples	
6	5	Perturbate tree	Seed fixed Base probability: 0.025, 0.05, 0.075, 0.1, 0.15, 0.2, 0.25, 0.3 Insert probability: 50 % Delete probability: 50 %
7	6	Process as vector	Identity class: Id^{P}, $\mathrm{Id}_2^{\mathrm{Pq}}$, $\mathrm{Id}_2^{\mathrm{PqOrder}}$, $\mathrm{Id}_{1,2}^{\mathrm{pq}}$, $\mathrm{Id}_{2,2}^{\mathrm{pq}}$, $\mathrm{Id}_2^{\mathrm{e,noise}}$, $\mathrm{Id}_2^{\mathrm{e,parent}}$ Similarity: $\langle\mathscr{T}\|1\|\mathscr{T}'\rangle$, $\langle\mathscr{T}\|M\|\mathscr{T}'\rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute, normalised distance and base data
8	5	Perturbate tree	*Compare step 6* Move probability: 100 %
9	8	Process as vector	*Compare step 7*
10	5	Perturbate tree	*Compuare step 6* Insert probability: 25 % Delete probability: 25 % Move probability: 50 %
11	10	Process as vector	*Compare step 7*
12	7, 9, 11	Calculate correlation	

Pseudo Metric Validation

The formalism introduced in this thesis targets the introduction of a pseudo metric. We cannot rely on exact metrics here, because we rely on approximating dynamic trees. Therefore we expect also different objects to have a distance of 0. This can happen for example when considering the similarity Id^{P} in the context of high repetition counts for vertices.

This workflow targets the analysis for meeting the requirements of pseudo metrics for the two implementations of SplittedStatistics: MultisetStatistics and incremental PDF statistics. For both methods we therefore evaluate metric characteristics of identity, symmetry, non-negativity as well as triangle inequality.

#	Input	Description	Configuration
1	–	Index by |vertices|	
2	1	Subset data	$\|\mathscr{T}\| > 1000$
3	2	Select data	Seed fixed Repetition: 1 Count: 1000
4	3	Process as matrix	Identity class: Id^{P} Similarity: $\langle \mathscr{T} \vert M \vert \mathscr{T} \rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute, normalised distance and base data
5	3	Process as matrix	*Compare step 4* incremental PDF statistics
6	4, 5	Analyse pseudo metric	

Splitted Statistics Analysis

For SplittedStatistics we currently implement two variants to represent attribute values:

- MultisetStatistics, and
- incremental PDF statistics.

This workflow is intended to analyse the characteristics of both approaches. We want to evaluate the precision regarding varied overlap of distributions and number of samples.

We already showed that MultisetStatistics is a pseudo-metric (see workflow B.2 on the previous page). Thus, it is symmetric but we can expect different pairs of trees to result in a distance of 0. The incremental PDF statistics does an approximation approach and therefore is not even a pseudo-metric.

Characteristics

To analyse the characteristics of MultisetStatistics and incremental PDF statistics we mainly focus the adaptation to changing distributions. We therefore first learn a given distribution for both statistics to apply a validation distribution to. The first characteristic to consider is the overlap: By iteratively changing the overlap of validation distribution from 100 % to 0 % we show the accuracy of both methods to an expected distance value.

Furthermore we are interested in varying numbers of samples. We therefore iteratively change the number of samples of the given validation distribution to show the influence. The number of samples is changed from 0 to a factor of 2. Again, for each variation the expected distance can be calculated and thus the deviation from expected distance.

#	Input	Description	Configuration
1	–	Analyse attribute statistics	

Deviation

This workflow is intended to analyse the *relative mean deviation error* regarding the weighting of attributes. I expect the deviation error to grow linearly with greater influence of attributes.

The workflow itself selects 100 randomly chosen trees. For each pair of trees, the distance matrix is calculated to analyse the deviation error. Results are plotted regarding the weighting.

#	Input	Description	Configuration
1	–	Index by |vertices|	
2	1	Subset data	$1000 < \|\mathscr{T}\| < 10\,000$
3	2	Select data	Seed fixed Repetition: 10 Count: 100
4	3	Process as matrix	Identity class: Id^{P} Similarity: $\langle \mathscr{T} \vert \{M, D\} \vert \mathscr{T}' \rangle$ Weighting: $\alpha_M : 0, \ldots, 1$, $\alpha_D : 1 - \alpha_M$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute, normalised distance and base data
5	3	Process as matrix	*Compare step 4* incremental PDF statistics
6	4, 5	Analyse deviation	

Sensitivity Analysis

The analysis of sensitivity focuses on determining the differences between high-quality vertices and low-quality vertices. High-quality vertices are vertices that have a high number of children and a big size of subtree of the specific vertex, Furthermore, high-quality vertices have attributes and a high number of attribute changes.

Therefore, we run two workflows that target the structure of trees as well as the attributes within a tree.

Structure

By definition from high-quality and low-quality vertices we therefore analyse distance results regarding perturbed trees for specific groups of vertices. The vertices we consider are inner vertices as high-quality vertices, and leaf vertices as low-quality vertices. We expect greater changes in distance for changes on high-quality vertices.

#	Input	Description	Configuration
1	-	Index by |vertices|	worker node = c01-007-125
2	1	Squish index	
3	2	Subset data	$25 < \|\mathscr{T}\| < 15\,000$
4	3	Aggregate data	
5	4	Select data	Seed fixed Count: 1 Repetition: 1
6	5	Perturbate tree	Seed fixed Repetition: 10 Base probability: (0.01, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2, 0.25) Insert probability: 50 % Delete probability: 50 % Leaf vertices only
7	6	Process as vector	Identity class: $\mathrm{Id}_2^{\mathrm{e,noise}}$ Similarity: $\langle\mathscr{T}\|1\|\mathscr{T}'\rangle$, $\langle\mathscr{T}\|M\|\mathscr{T}'\rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute, normalised distance and base data
8	6	Process as vector	*Compare step 7* Identity class: $\mathrm{Id}_3^{\mathrm{e,noise}}$
9	6	Process as vector	*Compare step 7* Identity class: $\mathrm{Id}_4^{\mathrm{e,noise}}$
10	5	Perturbate tree	*Compare step 6* Inner vertices only
11	10	Process as vector	*Compare step 7*
12	10	Process as vector	*Compare step 8*
13	10	Process as vector	*Compare step 9*
13	7, 8, 9, 11, 12, 13	Prepare Plot	

Attributes

High-quality vertices are not only related to vertices with a great amount of children but also to vertices having a great amount of attributes or even repeatedly changing attribute values. For this workflow we therefore measure the impacts on tree distances while removing different amounts of vertices containing attributes. For this analysis we expect the distance to increase with increasing number of attribute values that are related to a vertex that is deleted from a given tree.

#	Input	Description	Configuration
1	–	Index by |attributes|	worker node = c01-007-125
2	1	Squish index	
3	2	Subset data	|attribute events| > 5000
4	3	Aggregate data	
5	4	Select data	Seed fixed Count: 1 Repetition: 1
6	5	Perturbate tree	Seed fixed Repetition: 10 Base probability: (0.01, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2, 0.25) Delete probability: 100 % Attribute vertices only
7	6	Process as vector	Identity class: $\mathrm{Id}_2^{\mathrm{e,noise}}$ Similarity: $\langle \mathscr{T} \vert M \vert \mathscr{T}' \rangle$, $\langle \mathscr{T} \vert \{M, D, \Lambda\} \vert \mathscr{T}' \rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute, normalised distance and base data
8	6	Process as vector	*Compare step 7* Identity class: $\mathrm{Id}_3^{\mathrm{e,noise}}$
9	6	Process as vector	*Compare step 7* Identity class: $\mathrm{Id}_4^{\mathrm{e,noise}}$
10	7, 8, 9	Prepare plot	

Ensemble Validation

This thesis introduces the concept of ensembles into tree distance measurements. For this, we introduce two specific examples: an ensemble distance to handle noisy data being defined by permutations of vertices having the same parent as well as an ensemble supporting a partial matching of parent vertices. We expect the noise ensemble measure to be effective handling out-of-order events in streams as well as noise introduced by the operating system with regard to our considered use case. Furthermore we expect the partial matching ensemble to be efficient to handle data that is subject to changes over time and therefore needs to be considered non-stationary.

The following two workflows briefly analyse the characteristics of both introduced identity ensemble classes.

Noise

The ensemble that targets the handling of permutations of vertices having the same parent for some given range q includes two identity classes:

- $\mathrm{Id}^{\mathrm{Pq}}$ that uses infinite-length encoding in $\mathcal{P}$ dimension and finite-length encoding in $\mathcal{Q}$ dimension, and
- $\mathrm{Id}^{\mathrm{PqOrder}}$ that uses infinite-length encoding in $\mathcal{P}$ dimension and fixed-length encoding in $\mathcal{Q}$ dimension while ensuring ordered $\mathcal{Q}$ dimension.

To analyse this ensemble $\mathrm{Id}^{\mathrm{e,noise}}$ we select one specific tree. The selected tree is disturbed for different probabilities in the range $[0, 0.1]$. Each distortion is repeated several times for statistical relevance.

#	Input	Description	Configuration
1	-	Index by |vertices|	worker node = c01-007-106
2	1	Squish index	
3	2	Subset data	$1500 < \|\mathscr{T}\| < 3000$
4	3	Select data	Seed fixed Repetition: 1 Count: 1
5	4	Perturbate tree	Seed fixed Base probability: 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 Move probability: 100 %
6	5	Process as vector	Identity class: $\mathrm{Id}_2^{\mathrm{Pq}}$, $\mathrm{Id}_2^{\mathrm{PqOrder}}$, $\mathrm{Id}_2^{\mathrm{e,noise}}$ Similarity: $\langle \mathscr{T} \vert M \vert \mathscr{T}' \rangle$ MultisetStatistics: int(x) Analysis: absolute, normalised distance
7	6	Prepare plot	

Use Case

The ensemble that targets the partial matching of parent vertices includes two identity classes:

- Id^{P} that uses infinite-length encoding in $\mathcal{P}$ dimension, and
- $\mathrm{Id}^{\mathrm{pq}}$ that uses finite-length encoding in $\mathcal{P}$ dimension and fixed-length encoding in $\mathcal{Q}$ dimension.

To analyse this ensemble $\mathrm{Id}^{\mathrm{e,parent}}$ we select one specific tree. The selected tree is disturbed for different probabilities in the range $[0, 0.1]$. Each distortion is repeated several times for statistical relevance.

#	Input	Description	Configuration
1	-	Index by |vertices|	worker node = c01-007-106
2	1	Squish index	
3	2	Subset data	$1500 < \|\mathscr{T}\| < 3000$
4	3	Select data	Seed fixed Repetition: 1 Count: 1
5	4	Perturbate tree	Seed fixed Base probability: 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.125, 0.15, 0.175, 0.2, 0.225, 0.25, 0.275, 0.3, 0.325, 0.35, 0.375, 0.4, 0.425, 0.45, 0.475, 0.5 Move probability: 100 %
6	5	Process as vector	Identity class: $\mathrm{Id}_2^{\mathrm{P}}$, $\mathrm{Id}_{1,0}^{\mathrm{pq}}$, $\mathrm{Id}_1^{\mathrm{e,parent}}$ Similarity: $\langle \mathscr{T} \vert M \vert \mathscr{T}' \rangle$ MultisetStatistics: $\mathrm{int}(x)$ Analysis: absolute, normalised distance
7	6	Prepare plot	

Benchmarking

To show the efficiency and scalability of our proposed approach we realise a benchmarking to evaluate the theoretical analysis of time and space complexity based on performance and compression measurement. To measure performance and compression of our approach we utilise the reference implementation ASSESS.

Performance

The benchmarking workflow measures performance characteristics of distances and identity classes. For distances we differentiate

- $\langle \mathscr{T} | 1 | \mathscr{T}' \rangle$,
- $\langle \mathscr{T} | M | \mathscr{T}' \rangle$, and
- $\langle \mathscr{T} | \{M, D\} | \mathscr{T}' \rangle$.

To cover a sufficient amount of identity classes we differentiate

- independent encoding of available dimensions
  - dynamic pq-grams $\mathrm{Id}^{\mathrm{pq}}$ for different combinations of p and q as an example for finite-length encoding of the two dimensions $\mathcal{P}$ and $\mathcal{Q}$,
  - Id^{P} as an example for infinite-length encoding in $\mathcal{P}$ dimension,
  - $\mathrm{Id}^{\mathrm{Pq}}$ for different combinations of q as an example of a combination of infinite-length encoding in $\mathcal{P}$ dimension and finite-length encoding in $\mathcal{Q}$ dimension,
  - $\mathrm{Id}^{\mathrm{PqOrder}}$ for different combinations of q as a more complex example that builds on $\mathrm{Id}^{\mathrm{Pq}}$ but also considers a sorting of $\mathcal{Q}$ dimension,
  - $\mathrm{Id}^{\mathrm{e,noise}}$ as an example for ensembles,
- dependent encoding
  - $\mathrm{Id}^{\mathrm{P(q)}}$ for different combinations of q as an example of a combination of infinite-length encoding in $\mathcal{P}$ dimension and finite-length encoding in $\mathcal{Q}$ dimension.

This range of examples covers different possibilities of how to encode trees This, in turn, enables the comparison of performance regarding these different characteristics.

For the analysis we select a number of trees with increasing amount of vertices. This is repeated several times for each identified range of vertices.

#	Input	Description	Configuration
1	–	Index by |vertices|	Worker node: c01-007-102, c01-007-103
2	1	Squish index	
3	2	Subset data	$\lvert\mathscr{T}\rvert < 500\,000$
4	3	Select data	Seed fixed Repetition: 2 Count: 2
5	4	Process as vector	Identity class: Id^{P}, $\mathrm{Id}_i^{\mathrm{PqOrder}}$, $\mathrm{Id}_{i,i}^{\mathrm{pq}}$, Id_i^{Pq}, $\mathrm{Id}_i^{\mathrm{e,noise}}$, $\mathrm{Id}_i^{P(q)}$, $\forall i \in \{2,3,4\}$ Similarity: $\langle\mathscr{T}\rvert 1\rvert\mathscr{T}\rangle$, $\langle\mathscr{T}\rvert M\rvert\mathscr{T}'\rangle$, $\langle\mathscr{T}\rvert\{M,D\}\rvert\mathscr{T}'\rangle$ MultisetStatistics: $\mathrm{int}(x)$ Analysis: overall runtime, identity runtime, distance runtime, absolute distance, and base data
6	5	Prepare plot	

Compression

This workflow empirically evaluates space complexity for different identity classes. In specific, we distinguish between fixed-length and infinite-length encoding schemes as different complexities are expected.

For selected payloads we collect different statistics:

- number of vertices,
- height of a tree,
- size of the alphabet, and
- number of unique identities.

This workflow evaluates

- dynamic pq-grams $\mathrm{Id}^{\mathrm{pq}}$ as an example for finite-length encoding of the two dimensions $\mathcal{P}$ and $\mathcal{Q}$,
- Id^{P} as an example for infinite-length encoding in $\mathcal{P}$ dimension,
- $\mathrm{Id}^{\mathrm{Pq}}$ as an example of a combination of infinite-length encoding in $\mathcal{P}$ dimension and finite-length encoding in $\mathcal{Q}$ dimension, and
- $\mathrm{Id}^{\mathrm{PqOrder}}$ for different combinations of q as a more complex example that builds on $\mathrm{Id}^{\mathrm{Pq}}$ but also considers a sorting of $\mathcal{Q}$ dimension.

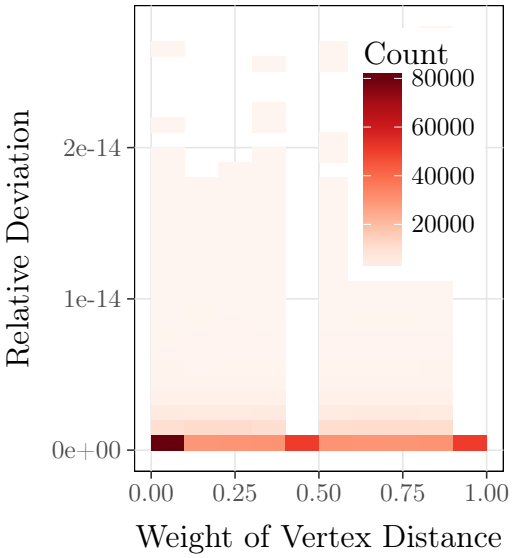
#	Input	Description	Configuration
1	–	Index by |vertices|	All worker nodes
2	1	Squish index	
3	2	Subset data	$\lvert\mathscr{T}\rvert > 25$
4	3	Select data	Seed fixed Repetition: 1 Count: 100
5	4	Analyse compression	Identity class: Id^{P}, $\mathrm{Id}_2^{\mathrm{PqOrder}}$, $\mathrm{Id}_{2,2}^{\mathrm{pq}}$, $\mathrm{Id}_2^{\mathrm{Pq}}$
6	5	Prepare plot	

Use Case Evalution

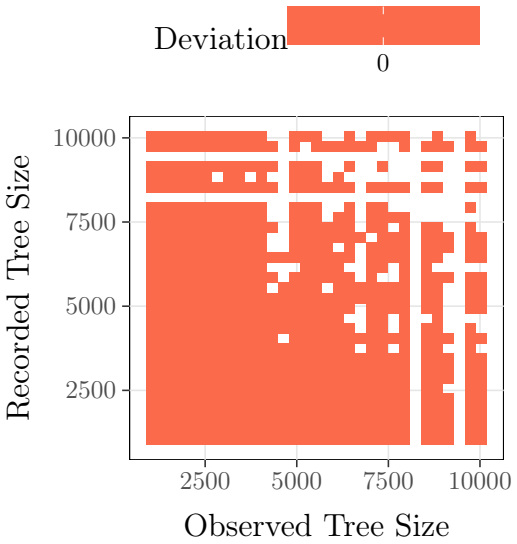
The final analysis of our proposed approach targets the validity of results regarding our considered use case of clustering, classification, and anomaly detection based on HEP batch jobs. We therefore consider different payloads from CMS collaboration that we mapped recorded job data from CERN Experiment Dashboard. The supplemented data enables the validation of clustering. Based on these results we make qualitative statements about the approach itself, its validity, and thus its feasibility for online analysis in production batch systems.

#	Input	Description	Configuration
1	–	Index by activity	All supplemented monitoring data
2	1	Subset data	*activity* $\in$ {pdmvserv_SMP-Summer12DR53X, alahiff_JME-Upg2023SHCAL14DR, vlimant_EGM-Fall14DR73, alahiff_HCA-Spring14dr}
3	2	Aggregate data	
4	3	Select data	Seed fixed Repetition: 1 Count: 1000
5	4	Process as matrix	Identity class: $\mathrm{Id}_2^{\mathrm{e,noise}}$ Similarity: $\langle\mathscr{T}\vert\{M,D,\Lambda\}\vert\mathscr{T}'\rangle$, $\langle\mathscr{T}\vert\{M,D\}\vert\mathscr{T}'\rangle$ MultisetStatistics: round($\sqrt{x}$) Analysis: absolute and normalised distance
6	5	Perform clustering	$\eta : 2, 3, \ldots, 10$ $\epsilon : 0.05, 0.1, 0.2, 0.3, 0.4, 0.5$
7	6	Perform classification	Identity class: $\mathrm{Id}_2^{\mathrm{e,noise}}$ Similarity: $\langle\mathscr{T}\vert\{M,D,\Lambda\}\vert\mathscr{T}'\rangle$ MultisetStatistics: round($\sqrt{(x)}$) Analysis: normalised distance, anomalies (threshold: 20 %)
8	7	Analyse classification	
9	7	Analyse anomaly detection	

C. Additional Plots

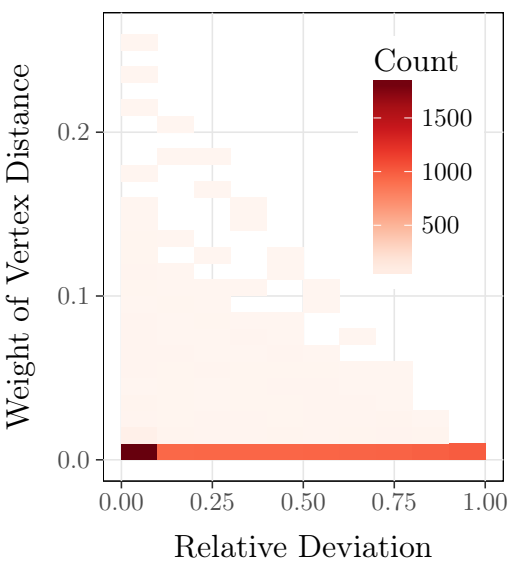


(a) Relative deviation e of distance result with regard to attribute influence

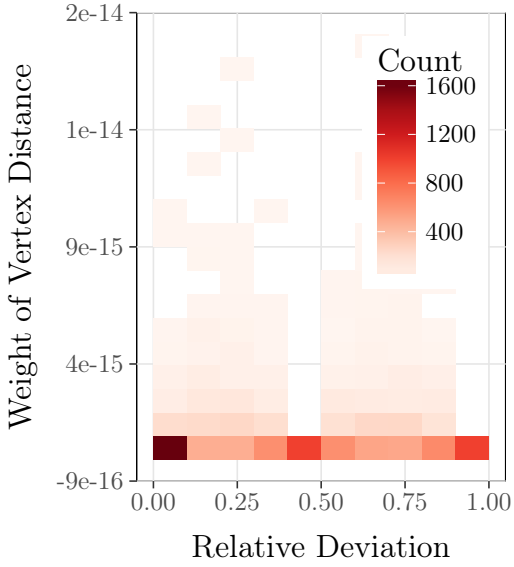


(b) Mean relative deviation $\bar{e}$ of distance result for different tree sizes due to attributes

Figure C.1.: Influence of attribute weighting and tree size to the relative deviation e for MultisetStatistics of calculated distance result for dynamic trees. Statistics have been calculated from 10 independent, randomly selected samples of 100 dynamic trees. The original dataset has been filtered to include only dynamic trees with sizes of $1000 < max(|\mathscr{T}_i|), \forall \mathscr{T}_i \in \mathscr{T} < 10\,000$. Figure C.1a shows influence of the weighting factor on the relative deviation e of the distance result. It can be seen, that deviation is 0 for weights 0, 0.5 and 1. As those weight map to distances of 2, 1 and 0, deviation e can only be an effect of floating point division errors. This effect can also be seen in Figure C.1b. Here, the mean relative deviation hardly differs throughout the range of tree sizes. Thus, the MultisetStatistics can be considered a pseudo metric.



(a) Relative deviation e of distance result for incremental PDF statistics



(b) Relative deviation e of distance result for MultisetStatistics

Figure C.2.: Relative deviation e of MultisetStatistics and incremental PDF statistics on diagonal

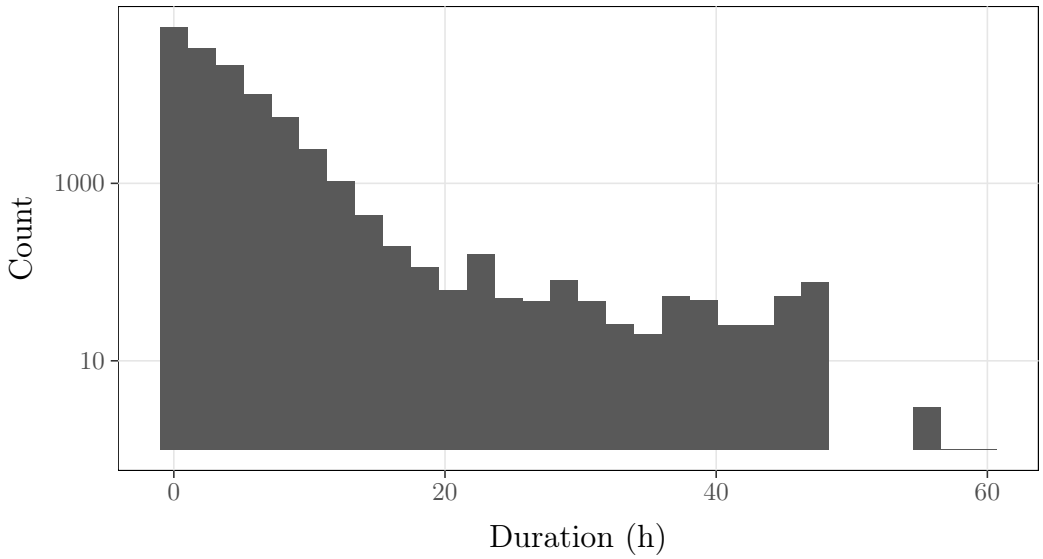


Figure C.3.: Visualisation of distribution of durations of CMS payloads.

D. Hardware and setup

At the GridKa data and computing centre the monitoring tool is currently running on two racks each consisting of 32 worker nodes. Each worker node has 16 physical cores with hyper-threading enabled. On each worker node 24 job slots are configured that are monitored in parallel. The operating system in use is Scientific Linux 6.4, a rebuild of Red Hat Enterprise Linux. Long term measurements and data collection are in progress with a measurement interval of 20 s and a logging of the complete process hierarchy for single batch jobs.

Each sensor agent gathers TCP/UDP packets and the corresponding batch job information for the assigned worker node. Extracted data is composed time series regarding the actual network traffic, relevant Unix process information as well as information about the batch job itself. These data is transmitted to a central collector and analysis component where specific traffic information per job are processed and analysed. As the job IDs from the batch system are not unique, a unique ID is generated. The central collector takes care of this by combining the start timestamp of the job with its job ID.

For an improved data handling and analysis the central component adds metadata to the measurements. They include general information about the monitoring process of a single job, e.g. duration, experiment, worker node, or additional information about possibly missing data. These are stored in a database enabling a fast access to specific batch job data.

Bibliography

[1] *A Multi-TeV Linear Collider based on CLIC Technology: CLIC Conceptual Design Report*. Tech. rep. Dec. 2013.

[2] HTCondor Admin. *HTCondor Homepage*. Feb. 2017. URL: `https://research.cs.wisc.edu/htcondor/` (visited on 02/21/2017).

[3] Anthony Agelastos et al. "Continuous whole-system monitoring toward rapid understanding of production HPC applications and systems." In: *Parallel Computing* 58 (2016), pp. 90–106.

[4] Anthony Agelastos et al. "The Lightweight Distributed Metric Service - A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications." In: *SC* (2014), pp. 154–165.

[5] Charu C Aggarwal. "Sketching Aggregates over Probabilistic Streams". In: *Managing and Mining Uncertain Data*. Boston, MA: Springer US, Mar. 2009, pp. 1–33.

[6] Cristina Aiftimiei et al. "GridICE: monitoring the user/application activities on the grid". In: *Journal of Physics: Conference Series* 119.6 (July 2008), p. 062003.

[7] Tatsuya Akutsu, Daiji Fukagawa, and Atsuhiro Takasu. "Approximating Tree Edit Distance through String Edit Distance." In: *Algorithmica* 57.2 (2010), pp. 325–348.

[8] Tatsuya Akutsu and Magnús M Halldórsson. "On the approximation of largest common subtrees and largest common point sets." In: *Theor. Comput. Sci.* 233.1-2 (2000), pp. 33–50.

[9] M Alishahi and M Naghibzadeh. "Tag name structure-based clustering of XML documents". In: *International Journal of ...* (2010).

[10] Mohamad Alishahi et al. "XML document clustering based on common tag names anywhere in the structure". In: *2009 14th International CSI Computer Conference (CSICC 2009) (Postponed from July 2009)*. IEEE, 2009, pp. 588–595.

[11] Mara Alpuente and Daniel Romero. "A Tool for Computing the Visual Similarity of Web Pages." In: *SAINT* (2010).

[12] Ganesh Ananthanarayanan et al. "Disk-Locality in Datacenter Computing Considered Irrelevant." In: *HotOS* (2011).

[13] P Andrade et al. "Service Availability Monitoring Framework Based On Commodity Software". In: *Journal of Physics: Conference Series* 396.3 (2012), p. 032008.

[14] J Andreeva et al. "Experiment Dashboard - a generic, scalable solution for monitoring of the LHC computing activities, distributed sites and services". In: *Journal of Physics: Conference Series* 396.3 (Dec. 2012), p. 032093.

[15] J Andreeva et al. "WLCG Transfers Dashboard: a Unified Monitoring Tool for Heterogeneous Data Transfers". In: *Journal of Physics: Conference Series* 513.3 (June 2014), p. 032005.

[16] F. Angiulli and C. Pizzuti. "Outlier mining in large high-dimensional data sets". In: *IEEE Transactions on Knowledge and Data Engineering* 17.2 (Feb. 2005), pp. 203–215. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.31.

[17] Tatsuya Asai et al. "Efficient algorithms for finding frequent substructures from semi-structured data streams". In: *JSAI'03/JSAI04: Proceedings of the 2003 and 2004 international conference on New frontiers in artificial intelligence*. Fujitsu. Springer-Verlag, June 2003, pp. 29–45.

[18] Tatsuya Asai et al. *Online algorithms for mining semi-structured data stream*. IEEE, 2002.

[19] Nikolaus Augsten and Michael Böhlen. "Similarity Joins in Relational Database Systems". In: *Synthesis Lectures on Data Management* 5.5 (Nov. 2013), pp. 1–124.

[20] Nikolaus Augsten, Michael Böhlen, and Johann Gamper. "An Incrementally Maintainable Index for Approximate Lookups in Hierarchical Data." In: *VLDB* (2006).

[21] Nikolaus Augsten, Michael Böhlen, and Johann Gamper. "Approximate Matching of Hierarchical Data Using pq-Grams." In: *VLDB* (2005), pp. 301–312.

[22] Nikolaus Augsten, Michael Böhlen, and Johann Gamper. "The pq -gram distance between ordered labeled trees". In: *ACM Transactions on Database Systems (TODS)* 35.1 (Feb. 2010), pp. 4–36.

[23] Nikolaus Augsten et al. "Approximate Joins for Data-Centric XML." In: *ICDE* (2008), pp. 814–823.

[24] Nikolaus Augsten et al. *On-the-fly token similarity joins in relational databases*. New York, New York, USA: ACM, June 2014.

[25] Nikolaus Augsten et al. "Windowed pq-grams for approximate joins of data-centric XML". In: *The VLDB Journal* 21.4 (Sept. 2011), pp. 463–488.

[26] Pablo Neira Ayuso, Rafael M Gasca, and Laurent Lefèvre. "Communicating between the kernel and user-space in Linux using Netlink sockets." In: *Softw., Pract. Exper.* 1 (2010), n/a–n/a.

[27] Imre Bárány and Van Vu. "Central limit theorems for Gaussian polytopes". In: *The Annals of Probability* 36.5 (Sept. 2008), pp. 1998–1998.

[28] Adam M Bates et al. "Trustworthy Whole-System Provenance for the Linux Kernel." In: *USENIX Security Symposium* (2015).

[29] Heinz Bauer. *Measure and Integration Theory*. Berlin, New York: DE GRUYTER, 2001.

[30] L A T Bauerdick and A Sciabà. "Towards a global monitoring system for CMS computing operations". In: *Journal of Physics: Conference Series* 396.3 (Dec. 2012), p. 032099.

[31] L Bauerdick et al. "Using Xrootd to Federate Regional Storage". In: *Journal of Physics: Conference Series* 396.4 (Dec. 2012), p. 042009.

[32] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. "A bag-of-features framework to classify time series". In: *IEEE transactions on pattern* (2013).

[33] Olaf Behnke et al., eds. *Data analysis in high energy physics*. Weinheim, Germany: Wiley-VCH, 2013. ISBN: 9783527410583, 9783527653447, 9783527653430. URL: http://www.wiley-vch.de/publish/dt/books/ISBN3-527-41058-9.

[34] Gerd Behrmann, Dmitry Ozerov, and Thomas Zangerl. "Xrootd in dCache - design and experiences". In: *Journal of Physics: Conference Series* 331.5 (2011), p. 052021.

[35] Gleb Beliakov, Ana Pradera, and Tomasa Calvo. *Aggregation Functions: A Guide for Practitioners*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 3540737200, 9783540737209.

[36] Douglas P Benjamin. "Grid Computing in the Collider Detector at Fermilab (CDF) scientific experiment". In: *CoRR* (2008).

[37] Karima Bessine et al. "XCLSC: Structure and content-based clustering of XML documents". In: *2015 12th International Symposium on Programming and Systems (ISPS)*. IEEE, 2015, pp. 1–7.

[38] Kevin S Beyer et al. "When Is "Nearest Neighbor" Meaningful?" In: *ICDT* 1540.Chapter 15 (1999), pp. 217–235.

[39] Abhinav Bhatele et al. "There Goes the Neighborhood: Performance Degradation due to Nearby Jobs". In: *the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, New York, USA: ACM Press, 2013, pp. 1–12.

[40] Albert Bifet et al. "New ensemble methods for evolving data streams". In: *the 15th ACM SIGKDD international conference*. New York, New York, USA: ACM Press, 2009, pp. 139–148.

[41] Philip Bille. "A survey on tree edit distance and related problems." In: *Theor. Comput. Sci.* 337.1-3 (2005), pp. 217–239.

[42] I Bird et al. *LHC Computing Grid*. Tech. rep. CERN-LHCC-2005-024.LCG-TDR-001. June 2005.

[43] I Bird et al. *Update of the Computing Models of the WLCG and the LHC Experiments*. Tech. rep. CERN-LHCC-2014-014. LCG-TDR-002. Apr. 2014.

[44] Kenneth Bloom and The CMS Collaboration. "CMS Use of a Data Federation". In: *Journal of Physics: Conference Series* 513.4 (2014), p. 042005.

[45] Hamed R Bonab and Fazli Can. "A Theoretical Framework on the Ideal Number of Classifiers for Online Ensembles in Data Streams." In: *CIKM* (2016), pp. 2053–2056.

[46] Daniele Bonacorsi. "CMS storage federations". In: *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference (2012 NSS/MIC)*. IEEE, 2012, pp. 2012–2015.

[47] Leo Breiman. "Bagging Predictors." In: *Machine Learning* (1996).

[48] Alex D Breslow et al. "The case for colocation of high performance computing workloads." In: *Concurrency and Computation - Practice and Experience* 28.2 (2016), pp. 232–251.

[49] David Buttler. "A Short Survey of Document Structure Similarity Algorithms." In: *International Conference on Internet Computing* (2004), pp. 3–9.

[50] Giulia Casarosa. "The Belle II Experiment". In: *Journal of Physics: Conference Series* 556.1 (Nov. 2014), p. 012072.

[51] Leopoldo Catania. "Dynamic Adaptive Mixture Models". In: *arXiv.org* (Mar. 2016). arXiv: `1603.01308v1 [stat.ME]`.

[52] CERN. *Future Linear Collider Study*. 2016. URL: https://fcc.web.cern.ch/ (visited on 03/17/2017).

[53] CERN. *Processing: What to record?* 2017. URL: https://home.cern/about/computing/processing-what-record (visited on 03/10/2017).

[54] CERN. *The Large Hadron Collider*. Jan. 2014. URL: http://home.cern/topics/large-hadron-collider (visited on 02/21/2017).

[55] José E Chacón. "Mixture model modal clustering". In: *arXiv.org* (Sept. 2016). arXiv: 1609.04721v1 [stat.ML].

[56] Edgar Chávez and Gonzalo Navarro. "A Probabilistic Spell for the Curse of Dimensionality." In: *ALENEX* 2153.Chapter 12 (2001), pp. 147–160.

[57] Sudarshan S Chawathe and Hector Garcia-Molina. "Meaningful change detection in structured data". In: *ACM SIGMOD Record* 26.2 (June 1997), pp. 26–37.

[58] Mostafa Haghir Chehreghani et al. "Clustering Rooted Ordered Trees". In: *2007 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2007, pp. 450–455.

[59] Abdur Chowdhury et al. "Collection statistics for fast duplicate document detection." In: *ACM Trans. Inf. Syst.* 20.2 (2002), pp. 171–191.

[60] Catalin Cirstoiu et al. "Monitoring, accounting and automated decision support for the alice experiment based on the MonALISA framework." In: *GMW@HPDC* (2007), p. 39.

[61] Sergio Cittolin, Attila Rácz, and Paris Sphicas. *CMS The TriDAS Project: Technical Design Report, Volume 2: Data Acquisition and High-Level Trigger. CMS trigger and data-acquisition project*. Technical Design Report CMS. Geneva: CERN, 2002. URL: http://cds.cern.ch/record/578006.

[62] ATLAS collaboration and G Aad. *The ATLAS Experiment at the CERN Large Hadron Collider, 2008*.

[63] the ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *arXiv.org* 1 (July 2012), pp. 1–29. arXiv: 1207.7214v2 [hep-ex].

[64] The CMS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *arXiv.org* 1 (July 2012), pp. 30–61. arXiv: 1207.7235v2 [hep-ex].

[65] COMPASS Collaboration and P Abbon. "The COMPASS Experiment at CERN". In: *arXiv.org* 3 (Mar. 2007), pp. 455–518. arXiv: hep-ex/0703049v1 [hep-ex].

[66] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844, 9780262033848.

[67] Graham Cormode and Minos N Garofalakis. "Sketching probabilistic data streams." In: *SIGMOD* (2007), pp. 281–292.

[68] Graham Cormode and S Muthukrishnan. "The string edit distance matching problem with moves." In: *ACM Trans. Algorithms* 3.1 (2007), p. 2.

[69] Univa Corporation. *Univa Corporation - Products Suite*. 2017. URL: http://www.univa.com/products/ (visited on 02/21/2017).

[70] G Costa and R Ortale. "On Effective XML Clustering by Path Commonality: An Efficient and Scalable Algorithm". In: *2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012)*. IEEE, 2012, pp. 389–396.

[71] Gianni Costa et al. "A Tree-Based Approach to Clustering XML Documents by Structure". In: *Knowledge Discovery in Databases: PKDD 2004*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, Sept. 2004, pp. 137–148.

[72] Isabel F Cruz et al. "Measuring Structural Similarity Among Web Documents - Preliminary Results." In: *EP* (1998).

[73] Theodore Dalamagas et al. "A methodology for clustering XML documents by structure." In: *Inf. Syst. ()* 31.3 (2006), pp. 187–228.

[74] Theodore Dalamagas et al. "Clustering XML Documents Using Structural Summaries." In: *EDBT Workshops* 3268.Chapter 54 (2004), pp. 547–556.

[75] Erik D Demaine et al. "An optimal decomposition algorithm for tree edit distance." In: *ACM Trans. Algorithms* 6.1 (2009), pp. 2–19.

[76] Thomas G Dietterich. "Ensemble Methods in Machine Learning". In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, June 2000, pp. 1–15.

[77] P A M Dirac. "A new notation for quantum mechanics". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 35.03 (Oct. 2008), pp. 416–418.

[78] Brendan Dolan-Gavitt et al. "Tappan Zee (North) Bridge: Mining Memory Accesses for Introspection". In: *the 2013 ACM SIGSAC conference*. New York, New York, USA: ACM Press, 2013, pp. 839–850.

[79] Laurence Field Domenico Giordano Cristovao Cordeiro. "CERN Computing in Commercial Clouds". 22nd International Conference on Computing in High Energy and Nuclear Physics. Oct. 2016. URL: `https://indico.cern.ch/event/505613/contributions/2227325/` (visited on 03/07/2017).

[80] Max Fischer Eileen Kuehn. *Python package index: dengraph*. 2017. URL: `https://pypi.python.org/pypi/dengraph` (visited on 03/13/2017).

[81] Elasticsearch. *Kibana: Explore, Visualize, Discover Data | Elastic*. 2017. URL: `https://www.elastic.co/products/kibana` (visited on 03/13/2017).

[82] M. Ellert et al. "The NorduGrid project: using Globus toolkit for building {GRID} infrastructure". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 502.23 (2003). Proceedings of the {VIII} International Workshop on Advanced Computing and Analysis Techniques in Physics Research, pp. 407–410. ISSN: 0168-9002. DOI: `https://doi.org/10.1016/S0168-9002(03)00453-4`. URL: `http://www.sciencedirect.com/science/article/pii/S0168900203004534`.

[83] Arnout Engelen. *NetHogs: Linux 'net top' tool*. 2017. URL: `https://github.com/raboof/nethogs` (visited on 03/02/2017).

[84] Günter Erli et al. "**On-demand provisioning of HEP compute resources on cloud sites and shared HPC centers** ". 22nd International Conference on Computing in High Energy and Nuclear Physics. Oct. 2016. URL: `https://indico.cern.ch/event/505613/contributions/2230729/` (visited on 03/07/2017).

[85] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.” In: *KDD* (1996).

[86] Tanja Falkowski, Anja Barth, and Myra Spiliopoulou. “DENGRAPH: A Density-based Community Detection Algorithm”. In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*. IEEE, 2007, pp. 112–115.

[87] L Field et al. “Towards sustainability: An interoperability outline for a Regional ARC based infrastructure in the WLCG and EGEE infrastructures”. In: *Journal of Physics: Conference Series* 219.6 (May 2010), p. 062051.

[88] Sergio Flesca et al. “Exploiting structural similarity for effective Web information extraction.” In: *Data Knowl. Eng. ()* 60.1 (2007), pp. 222–234.

[89] Sergio Flesca et al. “Fast Detection of XML Structural Similarity.” In: *IEEE Trans. Knowl. Data Eng. ()* 17.2 (2005), pp. 160–175.

[90] Ian T Foster, Carl Kesselman, and Steven Tuecke. “The Anatomy of the Grid - Enabling Scalable Virtual Organizations.” In: *IJHPCA* 15.3 (2001), pp. 200–222.

[91] Patrick Fuhrmann and Volker Gülzow. “dCache, Storage System for the Future.” In: *Euro-Par* 4128.Chapter 116 (2006), pp. 1106–1113.

[92] Robert Gardner et al. “Data federation strategies for ATLAS using XRootD”. In: *Journal of Physics: Conference Series* 513.4 (June 2014), p. 042049.

[93] Minos N Garofalakis and Amit Kumar. “Correlating XML data streams using tree-edit distance embeddings.” In: *PODS* (2003), pp. 143–154.

[94] Minos N Garofalakis and Amit Kumar. “XML stream processing using tree-edit distance embeddings.” In: *ACM Trans. Database Syst. ()* 30.1 (2005), pp. 279–332.

[95] ILC GDE. *ILC - International Linear Collider*. 2013. URL: `http://www.linearcollider.org/ILC` (visited on 02/23/2017).

[96] Andreas Gellrich. “Integration of grid and local batch system resources at DESY”. 22nd International Conference on Computing in High Energy and Nuclear Physics. Oct. 2016. URL: `https://indico.cern.ch/event/505613/contributions/2227414/` (visited on 03/07/2017).

[97] Pierre Geurts. “Pattern Extraction for Time Series Classification”. In: *Principles of Data Mining and Knowledge Discovery*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, Sept. 2001, pp. 115–127.

[98] Saptarshi Ghosh and Pabitra Mitra. “Combining content and structure similarity for XML document classification using composite SVM kernels”. In: *2008 19th International Conference on Pattern Recognition (ICPR)*. IEEE, 2008, pp. 1–4.

[99] Globus. *GT 6.0 GridFTP*. 2017. URL: `http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/` (visited on 02/27/2017).

[100] M Gollapalli et al. *Approximate Record Matching Using Hash Grams*. IEEE, 2011.

[101] Luis Guerra et al. “A comparison of clustering quality indices using outliers and noise.” In: *Intell. Data Anal.* (2012).

[102] Arvind Gupta and Naomi Nishimura. “Finding Largest Subtrees and Smallest Supertrees.” In: *Algorithmica* 21.2 (1998), pp. 183–210.

[103] M Hadjieleftheriou, J W Byers, and G Kollios. "Robust sketching and aggregation of distributed data streams". In: (2005).

[104] G Hamerly and C Elkan. "Learning the k in k-means". In: *NIPS* (2003).

[105] Andreas Heiss. *GridKa Monitoring Dashboard*. 2017. URL: `http://web-kit.gridka.de/monitoring/` (visited on 02/16/2017).

[106] Tim Henderson. *Tree edit distance using the Zhang Shasha algorithm*. 2014. URL: `https://pypi.python.org/pypi/zss/1.1.2` (visited on 03/13/2017).

[107] Israel Nathan Herstein. *Topics in algebra*. Tech. rep. New York, Toronto, London, 1964.

[108] KEK High Energy Accelerator Research Organization. *SuperKEKB Project*. 2011. URL: `http://www-superkekb.kek.jp` (visited on 02/21/2017).

[109] Kouichi Hirata, Yoshiyuki Yamamoto, and Tetsuji Kuboyama. "Improved MAX SNP-Hard Results for Finding an Edit Distance between Unordered Trees." In: *CPM* 6661.Chapter 34 (2011), pp. 402–415.

[110] D Hufnagel and the CMS Collaboration. "Enabling opportunistic resources for CMS Computing Operations". In: *Journal of Physics: Conference Series* 664.2 (Dec. 2015), p. 022025.

[111] Elena Ikonomovska, João Gama, and Sao Deroski. "Online tree-based ensembles and option trees for regression on evolving data streams". In: *Neurocomputing* 150 (Feb. 2015), pp. 458–470.

[112] Piotr Indyk. "Stable distributions, pseudorandom generators, embeddings, and data stream computation." In: *J. ACM ()* 53.3 (2006), pp. 307–323.

[113] Henry F Inman and Edwin L Bradley Jr. "The overlapping coefficient as a measure of agreement between probability distributions and point estimation of the overlap of two normal densities". In: *Communications in Statistics - Theory and Methods* 18.10 (June 2007), pp. 3851–3874.

[114] Intel Corporation. *Intelő 64 and IA-32 Architectures Software Developer's Manual*. Dec. 2016.

[115] Swami Iyer and Dan A. Simovici. "Structural classification of XML documents using multisets". In: *International Journal on Artificial Intelligence Tools* 17.05 (2008), pp. 1003–1022. DOI: `10.1142/S0218213008004266`. URL: `http://www.worldscientific.com/doi/abs/10.1142/S0218213008004266`.

[116] Anubhav Jain et al. "FireWorks - a dynamic workflow system designed for high-throughput applications." In: *Concurrency and Computation - Practice and Experience* 27.17 (2015), pp. 5037–5059.

[117] Youngseon Jeong, Myong Kee Jeong, and Olufemi A Omitaomu. "Weighted dynamic time warping for time series classification." In: *Pattern Recognition* 44.9 (2011), pp. 2231–2240.

[118] Yang Ji, Sangho Lee, and Wenke Lee. "RecProv - Towards Provenance-Aware User Space Record and Replay." In: *IPAW* 9672.Chapter 1 (2016), pp. 3–15.

[119] Bin Jiang et al. "Clustering Uncertain Data Based on Probability Distribution Similarity." In: *IEEE Trans. Knowl. Data Eng. ()* (2013).

[120] Haifeng Jiang et al. “Holistic Twig Joins on Indexed XML Documents.” In: *VLDB* (2003), pp. 273–284.

[121] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. “Alignment of trees an alternative to tree edit”. In: *Theoretical Computer Science* 143.1 (July 1995), pp. 137–148.

[122] A Jokanovic, J C Sancho, and G Rodriguez. “Quiet neighborhoods: Key to protect job performance predictability”. In: *(IPDPS)* (2015).

[123] Sachindra Joshi et al. “A bag of paths model for measuring structural similarity in Web documents.” In: *KDD* (2003), pp. 577–582.

[124] Anna Jurek et al. “A survey of commonly used ensemble-based classification techniques.” In: *Knowledge Eng. Review* 29.05 (2014), pp. 551–581.

[125] Karin Kailing et al. “Efficient Similarity Search for Hierarchical Data in Large Databases.” In: *EDBT* 2992.Chapter 39 (2004), pp. 676–693.

[126] K Kailing et al. “Efficient similarity search in large databases of tree structured objects”. In: *Proceedings. 20th International Conference on Data Engineering*. IEEE, 2004, p. 835.

[127] Melanie Kambadur et al. “Measuring interference between live datacenter applications”. In: *2012 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–12.

[128] Vasileios P Kemerlis et al. “libdft - practical dynamic data flow tracking for commodity systems.” In: *VEE* 47.7 (2012), pp. 121–132.

[129] E Keogh. *Data mining and machine learning in time series databases*. Tutorial in ICML, 2004.

[130] Eamonn J Keogh and Shruti Kasetty. “On the Need for Time Series Data Mining Benchmarks - A Survey and Empirical Demonstration.” In: *Data Min. Knowl. Discov.* 7.4 (2003), pp. 349–371.

[131] Eamonn Keogh and Chotirat Ann Ratanamahatana. “Exact indexing of dynamic time warping”. In: *Knowledge and Information Systems* 7.3 (Mar. 2005), pp. 358–386.

[132] Michael Kerrisk. *cgroups - Linux control groups*. 2016. URL: `http://man7.org/linux/man-pages/man7/cgroups.7.html` (visited on 02/20/2017).

[133] Michael Kerrisk. *ptrace - Linux manual page*. 2016. URL: `http://man7.org/linux/man-pages/man2/ptrace.2.html` (visited on 02/16/2017).

[134] Michael Kerrisk. *strace - Linux manual page*. 2016. URL: `http://man7.org/linux/man-pages/man1/strace.1.html` (visited on 02/16/2017).

[135] Samuel T King and Peter M Chen. “Backtracking intrusions”. In: *ACM Transactions on Computer Systems* 23.1 (Feb. 2005), pp. 51–76.

[136] Andreas Knüpfer et al. “The Vampir Performance Analysis Tool-Set”. In: *Tools for High Performance Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 139–155.

[137] Eileen Kuehn. “Clustering Evolving Batch System Jobs for Online Anomaly Detection”. In: *ICDMW ’15: Proceedings of the 2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE Computer Society, Nov. 2015, pp. 1534–1535.

[138] Eileen Kuehn and Achim Streit. "Online Distance Measurement for Tree Data Event Streams." In: *DASC/PiCom/DataCom/CyberSciTech* (2016), pp. 681–688.

[139] Eileen Kuehn et al. "Active Job Monitoring in Pilots". In: *Journal of Physics: Conference Series* 664.5 (Dec. 2015), p. 052019.

[140] Eileen Kuehn et al. "Analyzing data flows of WLCG jobs at batch job level". In: *Journal of Physics: Conference Series* 608.1 (May 2015), p. 012017.

[141] Eileen Kuehn et al. "Monitoring Data Streams at Process Level in Scientific Big Data Batch Clusters." In: *BDC* (2014), pp. 90–95.

[142] E Kuehn et al. "A scalable architecture for online anomaly detection of WLCG batch jobs". In: *Journal of Physics: Conference Series* 762.1 (Nov. 2016), p. 012002.

[143] Sangeetha Kutty, Richi Nayak, and Yuefeng Li. *HCX: an efficient hybrid clustering approach for XML documents*. an efficient hybrid clustering approach for XML documents. New York, New York, USA: ACM, Sept. 2009.

[144] Issam H Laradji, Mohammed Salahadin, and Lahouari Ghouti. "XML classification using ensemble learning on extracted features." In: *ACM Southeast Regional Conference* (2014), pp. 1–6.

[145] Thomas Larsson and Tomas Akenine-Möller. "Collision Detection for Continuously Deforming Bodies". In: *Eurographics 2001 - Short Presentations*. Eurographics Association, 2001. DOI: 10.2312/egs.20011005.

[146] Chris Lattner and Vikram Adve. *LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation*. IEEE Computer Society, Mar. 2004.

[147] Aleksandar Lazarevic et al. "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection". In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. Philadelphia, PA: Society for Industrial and Applied Mathematics, Dec. 2013, pp. 25–36.

[148] Mong-Li Lee et al. "XClust: clustering XML schemas for effective integration." In: *CIKM* (2002), pp. 292–299.

[149] Anna Lesniewska. "Clustering XML Documents by Structure." In: *ADBIS* 5968.Chapter 30 (2009), pp. 238–246.

[150] Fei Li et al. "A survey on tree edit distance lower bound estimation techniques for similarity join on XML data". In: *ACM SIGMOD Record* 42.4 (Feb. 2014), pp. 29–39.

[151] Wang Lian et al. "An Efficient and Scalable Algorithm for Clustering XML Documents by Structure." In: *IEEE Trans. Knowl. Data Eng. ()* 16.1 (2004), pp. 82–96.

[152] Jessica Lin, Rohan Khade, and Yuan Li. "Rotation-invariant similarity in time series using bag-of-patterns representation". In: *Journal of Intelligent Information Systems* 39.2 (2012), pp. 287–315.

[153] Jessica Lin and Yuan Li. "Finding Structural Similarity in Time Series Data Using Bag-of-Patterns Representation." In: *SSDBM* 5566.Chapter 33 (2009), pp. 461–477.

[154] J Lin et al. "Experiencing SAX: a novel symbolic representation of time series". In: *Data Mining and Knowledge Discovery* (2007).

[155] Jason Lines and Anthony Bagnall. "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29.3 (2015), pp. 565–592.

[156] Lei Liu et al. “A Methodology for Clustering XML Documents Based on Labeled Tree”. In: *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE, 2009, pp. 397–401.

[157] Markus Lorch et al. “Authorization and account management in the Open Science Grid.” In: *GRID* (2005).

[158] Chi-Keung Luk et al. “Pin - building customized program analysis tools with dynamic instrumentation.” In: *PLDI* 40.6 (2005), pp. 190–200.

[159] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2009.

[160] Matthew L Massie, Brent N Chun, and David E Culler. “The ganglia distributed monitoring system - design, implementation, and experience.” In: *Parallel Computing* 30.7 (2004), pp. 817–840.

[161] B T Meadows. “The BaBar Experiment at SLAC”. In: *Physics of Mass*. Boston: Kluwer Academic Publishers, 2002, pp. 227–236.

[162] Bernd Mohr et al. “The HOPSA Workflow and Tools”. In: *Tools for High Performance Computing 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, May 2013, pp. 127–146.

[163] S Muthukrishnan and Süleyman Cenk Sahinalp. “Approximate nearest neighbors and sequence comparison with block operations.” In: *STOC* (2000), pp. 416–424.

[164] Javier Navaridas, Jose Antonio Pascual, and José Miguel-Alonso. “Effects of Job and Task Placement on Parallel Scientific Applications Performance.” In: *PDP* (2009).

[165] Richi Nayak. “Fast and effective clustering of XML data using structural information.” In: *Knowl. Inf. Syst.* (2008).

[166] Richi Nayak and Wina Iryadi. “XML schema clustering with semantic and hierarchical similarity measures.” In: *Knowl.-Based Syst. ()* 20.4 (2007), pp. 336–349.

[167] Andrew Nierman and H V Jagadish. “Evaluating Structural Similarity in XML Documents.” In: *WebDB* (2002), pp. 61–66.

[168] P Nilsson et al. “Next Generation PanDA Pilot for ATLAS and Other Experiments”. In: *Journal of Physics: Conference Series* 513.3 (June 2014), p. 032071.

[169] Torkel Ödegaard and Raintank Inc. *Grafana - Beautiful Metrics Dashboards, Data Visualization and Monitoring*. 2015. URL: `http://grafana.org` (visited on 03/13/2017).

[170] LCG Office. *REBUS: Topology - Federation pledges for DE-KIT in year 2016*. 2017. URL: `https://wlcg-rebus.cern.ch/apps/topology/federation/211/` (visited on 02/27/2017).

[171] WLCG Project Office. *Welcome to the Worldwide LHC Computing Grid | WLCG*. 2016. URL: `http://wlcg.web.cern.ch` (visited on 02/21/2017).

[172] M de Palma. “The CMS experiment at LHC”. In: *Nuclear Physics B - Proceedings Supplements* 61.3 (Feb. 1998), pp. 32–38.

[173] Ady Wahyudi Paundu et al. “Leveraging Static Probe Instrumentation for VM-based Anomaly Detection System.” In: *ICICS* 9543.Chapter 27 (2015), pp. 320–334.

[174] Mateusz Pawlik and Nikolaus Augsten. “A Memory-Efficient Tree Edit Distance Algorithm.” In: *DEXA* 8644.Chapter 16 (2014), pp. 196–210.

[175] Andrea Perrotta. "Performance of the CMS High Level Trigger". In: *Journal of Physics: Conference Series* 664.8 (Dec. 2015), p. 082044.

[176] Jerome Petazzonl. *Gathering LXC and Docker Containers Metrics*. 2013. URL: `https://blog.docker.com/2013/10/gathering-lxc-docker-containers-metrics/` (visited on 02/20/2017).

[177] Maciej Piernik, Dariusz Brzezinski, and Tadeusz Morzy. "Clustering XML documents by patterns". In: *Knowledge and Information Systems* 46.1 (2016), pp. 185–212.

[178] Maciej Piernik et al. "XML clustering: a review of structural approaches". In: *The Knowledge Engineering Review* 30.03 (May 2015), pp. 297–323.

[179] Devin J Pohly et al. "Hi-Fi - collecting high-fidelity whole-system provenance." In: *ACSAC* (2012), pp. 259–268.

[180] The Icinga Project. *Icinga - Open Source Monitoring*. 2017. URL: `https://www.icinga.com` (visited on 02/21/2017).

[181] Hong-Jun Qiu and Wen-Jing Yu. "A methodology for using edges to measure structural and semantic similarity of XML documents". In: *2009 International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, 2009, pp. 1653–1658.

[182] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient Algorithms for Mining Outliers from Large Data Sets." In: *SIGMOD Conference* 29.2 (2000), pp. 427–438.

[183] Chotirat Ann Ratanamahatana and Eamonn J Keogh. "Making Time-Series Classification More Accurate Using Learned Constraints." In: *SDM* (2004), pp. 11–22.

[184] Chotirat Ann Ratanamahatana and Eamonn J Keogh. "Three Myths about Dynamic Time Warping Data Mining." In: *SDM* (2005), pp. 506–510.

[185] Gang Ren et al. "Google-Wide Profiling - A Continuous Profiling Infrastructure for Data Centers." In: *IEEE Micro* (2010).

[186] Leonardo Ribeiro and Theo Härder. "Evaluating Performance and Quality of XML-Based Similarity Joins." In: *ADBIS* 5207.Chapter 18 (2008), pp. 246–261.

[187] Juan José Rodrguez, Carlos J Alonso, and José A Maestro. "Support vector machines of interval-based features for time series classification." In: *Knowl.-Based Syst. ()* 18.4-5 (2005), pp. 171–178.

[188] Frédéric Ros and Serge Guillaume. "DENDIS - A new density-based sampling for clustering algorithm." In: *Expert Syst. Appl.* 56 (2016), pp. 349–359.

[189] Tai Sakuma and Thomas McCauley. "Detector and Event Visualization with SketchUp at the CMS Experiment". In: *Journal of Physics: Conference Series* 513.2 (2014), p. 022032. URL: `http://stacks.iop.org/1742-6596/513/i=2/a=022032`.

[190] Till Schäfer and Petra Mutzel. "StruClus - Structural Clustering of Large-Scale Graph Databases." In: *CoRR* cs.DB (2016).

[191] Robert E Schapire and Yoram Singer. "Improved Boosting Algorithms Using Confidence-rated Predictions." In: *Machine Learning* 37.3 (1999), pp. 297–336.

[192] Nico Schlitter, Tanja Falkowski, and Jörg Lässig. "DenGraph-HO - a density-based hierarchical graph clustering algorithm." In: *Expert Systems* 31.5 (2014), pp. 469–479.

[193] Nico Schlitter, Tanja Falkowski, and Jörg Lässig. "DenGraph-HO - Density-based Hierarchical Community Detection for Explorative Visual Network Analysis." In: *SGAI Conf.* Chapter 22 (2011), pp. 283–296.

[194] Matthias Schnepf. "Calculation of cross-section limits for the production of single top quarks in association with a Higgs boson using container technologies". MS. Karlsruhe Institute of Technology, 2017. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48876.

[195] J Schovancova et al. "ATLAS Distributed Computing Monitoring tools during the LHC Run I". In: *Journal of Physics: Conference Series* 513.3 (June 2014), p. 032084.

[196] J Schukraft. "The ALICE heavy-ion experiment at the CERN LHC". In: *Nuclear Physics A* 566 (Jan. 1994), pp. 311–319.

[197] Rolf Seuster et al. "Context-aweare distributed cloud computing using CloudScheduler". 22nd International Conference on Computing in High Energy and Nuclear Physics. Oct. 2016. URL: https://indico.cern.ch/event/505613/contributions/2230405/ (visited on 03/07/2017).

[198] Igor Sfiligoi et al. "The Pilot Way to Grid Resources Using glideinWMS". In: *2009 WRI World Congress on Computer Science and Information Engineering*. IEEE, 2009, pp. 428–432.

[199] Dana Shapira and James A Storer. "Edit distance with move operations". In: *Journal of Discrete Algorithms* 5.2 (June 2007), pp. 380–392.

[200] Hadi Sharifi, Omar Aaziz, and Jonathan Cook. "Monitoring HPC applications in the production environment". In: *the 2nd Workshop*. New York, New York, USA: ACM Press, 2015, pp. 39–47.

[201] Li Sheng en et al. "An Efficient Semantic Similarity Search on XML Documents". In: *2010 International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2010, pp. 86–90.

[202] Jamie Shiers. "The Worldwide LHC Computing Grid (worldwide LCG)." In: *Computer Physics Communications* 177.1-2 (2007), pp. 219–223.

[203] Arie Shoshani, Alexander Sim, and Junmin Gu. "Storage Resource Managers". In: *Grid Resource Management: State of the Art and Future Trends*. Ed. by Jarek Nabrzyski, Jennifer M. Schopf, and Jan Wglarz. Boston, MA: Springer US, 2004, pp. 321–340. ISBN: 978-1-4615-0509-9. DOI: 10.1007/978-1-4615-0509-9_20. URL: http://dx.doi.org/10.1007/978-1-4615-0509-9_20.

[204] M. Stamatogiannakis, P.T. Groth, and H.J. Bos. "Decoupling Provenance Capture and Analysis from Execution". In: *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP'15)*. 2015.

[205] Manolis Stamatogiannakis, Paul T Groth, and Herbert Bos. "Looking Inside the Black-Box - Capturing Data Provenance Using Dynamic Instrumentation." In: *IPAW* 8628.Chapter 12 (2014), pp. 155–167.

[206] D E Sturim et al. "Speaker verification using text-constrained Gaussian Mixture Models". In: *Proceedings of ICASSP '02*. IEEE, 2002, pp. I–677–I–680.

[207] Andrea Tagarelli, Mario Longo, and Sergio Greco. "Word Sense Disambiguation for XML Structure Feature Generation." In: *ESWC* (2009).

[208] Kamal Taha and Ramez Elmasri. "XCDSearch: An XML Context-Driven Search Engine". In: *IEEE Transactions on Knowledge and Data Engineering* 22.12 (2010), pp. 1781–1796.

[209] Kuo-Chung Tai. "The Tree-to-Tree Correction Problem." In: *J. ACM ()* 26.3 (1979), pp. 422–433.

[210] Yoshimasa Takabatake, Yasuo Tabei, and Hiroshi Sakamoto. "Online Pattern Matching for String Edit Distance with Moves." In: *SPIRE* 8799.Chapter 20 (2014), pp. 203–214.

[211] Naiyana Tansalarak and Kajal T Claypool. "QMatch - Using paths to match XML schemas." In: *Data Knowl. Eng. ()* 60.2 (2007), pp. 260–282.

[212] Yufei Tao, Xiaokui Xiao, and Shuigeng Zhou. "Mining distance-based outliers from large databases in any metric space". In: *the 12th ACM SIGKDD international conference*. New York, New York, USA: ACM Press, 2006, p. 394.

[213] Dawood Tariq, Maisem Ali, and Ashish Gehani. "Towards Automated Collection of Application-Level Data Provenance." In: *TaPP* (2012).

[214] Shirish Tatikonda and Srinivasan Parthasarathy. "Hashing tree-structured data: Methods and applications." In: *ICDE* (2010), pp. 429–440.

[215] tcpdump. *TCPDUMP/LIBPCAP public repository*. 2013. URL: `http://www.tcpdump.org` (visited on 03/13/2017).

[216] Joe Tekli. "An Overview on XML Semantic Disambiguation from Unstructured Text to Semi-Structured Data: Background, Applications, and Ongoing Challenges". In: *IEEE Transactions on Knowledge and Data Engineering* PP.99 (2016), pp. 1–1.

[217] Inc. The Cacti Group. *Cacti - The Complete RRDTool-based Graphing Solution*. 2017. URL: `http://www.cacti.net` (visited on 02/21/2017).

[218] The CMS Trigger and Data Acquisition Group. "The CMS High Level Trigger". In: *arXiv.org* 3 (Dec. 2005), pp. 605–667. arXiv: `hep-ex/0512077v1 [hep-ex]`.

[219] *The International Linear Collider - Technical Design Report*. Tech. rep. 2013.

[220] The LHCb Collaboration et al. " The LHCb Detector at the LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08005–S08005.

[221] M Theobald, R Schenkel, and G Weikum. "Exploiting Structure, Annotation, and Ontological Knowledge for Automatic Classification of XML Data." In: *WebDB* (2003).

[222] M Tomasek, M Cajkovsky, and I Klimek. "Cloud-centric application tracing and user monitoring intrusion prevention system". In: *2013 IEEE 17th International Conference on Intelligent Engineering Systems (INES)*. IEEE, 2013, pp. 339–343.

[223] Tien Tran, Richi Nayak, and Peter Bruza. "Combining Structure and Content Similarities for XML Document Clustering." In: *AusDM* (2008), pp. 219–225.

[224] Ken Ueno et al. "Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining." In: *ICDM* (2006).

[225] Jonas Wagner et al. "High System-Code Security with Low Overhead". In: *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2015, pp. 866–879.

[226] Xiaozhe Wang, Kate A Smith, and Rob J Hyndman. "Characteristic-Based Clustering for Time Series Data." In: *Data Min. Knowl. Discov.* 13.3 (2006), pp. 335–364.

[227] WLCG Project Office. *Tiers (as at June 2014)*. 2015. URL: https://espace2013.cern.ch/WLCG-document-repository/images1/WLCG/WLCG-TiersJun14_v9.png (visited on 01/04/2017).

[228] Hao Wu et al. "Automatic Cloud Bursting under FermiCloud". In: *2013 International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2013, pp. 681–686.

[229] Xin Wu and Guiquan Liu. "XML Twig Pattern Matching Using Version Tree". In: *Data Knowl. Eng.* 64.3 (Mar. 2008), pp. 580–599. ISSN: 0169-023X. DOI: 10.1016/j.datak.2007.09.013. URL: http://dx.doi.org/10.1016/j.datak.2007.09.013.

[230] Zhengzheng Xing, Jian Pei, and Philip S Yu. "Early Prediction on Time Series - A Nearest Neighbor Approach." In: *IJCAI* (2009).

[231] J Xu, A H Sung, and Q Liu. "Tree based behavior monitoring for adaptive fraud detection". In: *Pattern Recognition* 1 (2006), pp. 1208–1211.

[232] R Xu and D WunschII. "Survey of Clustering Algorithms". In: *IEEE Transactions on neural networks* 16.3 (May 2005), pp. 645–678.

[233] Rui Yang, Panos Kalnis, and Anthony K. H. Tung. "Similarity Evaluation on Tree-structured Data". In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. SIGMOD '05. Baltimore, Maryland: ACM, 2005, pp. 754–765. ISBN: 1-59593-060-4. DOI: 10.1145/1066157.1066243. URL: http://doi.acm.org/10.1145/1066157.1066243.

[234] Shanzhen Yi, Bo Huang, and Weng Tat Chan. "XML Application Schema Matching Using Similarity Measure and Relaxation Labeling". In: *Inf. Sci.* 169.1-2 (Jan. 2005), pp. 27–46. ISSN: 0020-0255. DOI: 10.1016/j.ins.2004.02.013. URL: http://dx.doi.org/10.1016/j.ins.2004.02.013.

[235] Heng Yin et al. "Panorama - capturing system-wide information flow for malware detection and analysis." In: *ACM Conference on Computer and Communications Security* (2007), pp. 116–127.

[236] Guo Yongming, Chen Dehua, and Le Jiajin. "Clustering XML Documents by Combining Content and Structure". In: *2008 International Symposium on Information Science and Engineering (ISISE)*. IEEE, 2008, pp. 583–587.

[237] J P Yoon, V Raghavan, and V Chakilam. "BitCube: a three-dimensional bitmap indexing for XML documents". In: *Thirteenth International Conference on Scientific and Statistical Database Management. SSDBM 2001*. IEEE Comput. Soc, pp. 158–167.

[238] Derek S Young. "An Overview of Mixture Models". In: *arXiv.org* (Aug. 2008). arXiv: 0808.0383v3 [math.ST].

[239] K. Zhang and D. Shasha. "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems". In: *SIAM J. Comput.* 18.6 (Dec. 1989), pp. 1245–1262. ISSN: 0097-5397. DOI: 10.1137/0218082. URL: http://dx.doi.org/10.1137/0218082.

[240] Kaizhong Zhang, Rick Statman, and Dennis Shasha. "On the Editing Distance Between Unordered Labeled Trees". In: *Inf. Process. Lett.* 42.3 (May 1992), pp. 133–139. ISSN: 0020-0190. DOI: 10.1016/0020-0190(92)90136-J. URL: http://dx.doi.org/10.1016/0020-0190(92)90136-J.

[241] Zhongping Zhang, Rong Li Shunliang Cao, and Yangyong Zhu. "Similarity metric for XML documents". In: *In Proc. of Workshop on Knowledge and Experience Management*. 2003.

[242] David Yu Zhu et al. "TaintEraser: Protecting Sensitive Data Leaks Using Application-Level Taint Tracking". In: *ACM SIGOPS Operating Systems Review* 45.1 (Feb. 2011), pp. 142–154.

Glossary

aggregation function A function aggregating distances from projecting multiple identities to a single distance. Ensembles of identities require an aggregation function to define how individual identities are combined. 107

Algorithm Simulation for Streaming Environments to aSsess tree Similarities Prototypical implementation of the distance measurement framework for dynamic and static trees proposed in this thesis. The prototype considers all relevant components including different identity classes and identity ensemble classes defined upon the identity classes, different distance and similarity functions, and implementations for MultisetStatistics as well as incremental PDF statistics. 129, 185

atomic vertex A fundamental vertex of a tree, composed only of a label and value. Atomic vertices never have children, and only occur as children of composite vertices. 82–87, 100, 181

Batch Process Network Monitor Prototypical implementation of a network monitoring sensor utilising libpcap as well as netsockets to acquire the required information of processes and network traffic. The implementation has been deployed in production environments and supports several flavours of the Linux distribution. 32, 185

Cluster Representative A compact representation of a cluster of entities. Simple Cluster Representatives may be formed as the average of entities of the cluster. For arbitrarily shaped clusters or non-Euclidean space, complex Cluster Representatives are required. 119–121, 185

composite vertex A complex vertex of a tree, with attributes and features. Composite vertices form fully featured trees, with each composite vertex potentially having a parent and multiple children. All features of a composite vertex can be expressed by atomic vertices as children. 80–87, 98, 100, 181

diamond A distortion in tree representations that attribute multiple parents to the same vertex. Lossy tree representations, such as embeddings, may retain enough information of a tree to differentiate parents, but not children. This causes separate branches in the representation to collapse, forming a diamond shaped structure. xi, 59–65, 108

dynamic tree event stream representation Extension of the tree event stream representation. The dynamic tree event stream representation supports all available tree edit operations to describe trees based on vertex events. 53, 54, 72

embedding A representation of trees with a simpler structure, such as vectors or sets. Commonly used with decomposition, storing representations of vertices or subtrees, or summaries, storing general features such as width, height, and fanout. 46, 181, 182

HTCondor Batch system for high throughput computing. HTCondor is designed to use spatially and administratively distributed resources. As such, it is commonly used for managing grid, cloud, and other opportunistic resources. 16

identity A fixed set of information shared by similar vertices of trees. An identity may encode features of a vertex, its position in a tree, or even related vertices such as parents, siblings or children. All vertices with the same identity are considered equivalent, and the identity may represent them equally. xv, 4, 50, 51, 54–73, 75, 80–82, 84–90, 92, 96–100, 102–111, 115, 116, 120, 121, 123, 124, 127, 131, 133, 136–138, 151, 153, 154, 161, 173, 181, 182

identity class A definition of the information to be included in the identity of each vertex in a tree. An identity class allows to create comparable identities for the vertices of one or multiple trees. The granularity and meaning of equality of vertices given their identities is defined by the identity class used to derive the identities. 55, 57, 63–66, 70, 72, 75, 82, 84, 85, 101–111, 115, 120, 129, 130, 134–140, 148, 153, 154, 161–165, 167–175, 182

identity ensemble A collection of multiple identities describing the same vertex. Even for the same vertex, each identity is separate. Comparing identity ensembles allows for partial equality if only a fraction of identities of vertices match. 102–111, 115, 116, 123, 182

identity ensemble class A group of identity classes defining the information for each identities of an identity ensemble. Implicitly defines the granularity and scope of full and partial identity equality. 101, 108, 138, 170

identity ensemble profile An identity profile composed of identity ensembles instead of identities. 103–105, 115, 116

identity ensemble profile projection An identity profile projection based on identity ensembles instead of identities. 106, 107

identity profile A collection of identities of each vertex of a single tree. An identity profile is an abstract representation of a tree, similar to how an embedding is an actual representation of a tree. 2, 50, 51, 57, 58, 67–75, 80, 81, 87, 93, 98, 99, 102–104, 107, 108, 115, 116, 118–121, 123, 124, 126, 151, 156, 182

identity profile projection A projection of one identity profile onto another, yielding the overlap of the two identity profiles. 68–72, 75, 81, 82, 85–88, 98, 102, 104, 106–108, 123, 139, 145, 182

identity profile projection operator An operator defining how the projection of identity profiles is performed to derive similarities and distances. The identity profile projection operator provides defines the space in which identity profiles are stored and compared. For example, an operator for multiplicity of identities uses the L^1-space for projections. xi, 69–72, 74, 84–88, 98, 99, 115, 120, 123, 136, 137, 139, 144, 145, 148

opportunistic resource Computing resources that are not permanently provided, but instead dynamically acquired. Usually refers only to processing resources, especially

from cloud providers. Resources are acquired only transiently when demand strongly outweighs static supply, and released promptly when demand declines. 12, 13, 20, 21, 31, 129

payload A payload is a user-defined batch job that runs within a pilot. Neither management nor scheduling of payloads are performed by the local batch system, but by the pilot and an external global batch job scheduler. xi, 3, 11, 12, 18, 19, 21–23, 26, 28–31, 34–38, 109, 130, 131, 140–143, 147, 148, 158–160, 173, 175, 178, 182

pilot A pilot is a special placeholder batch job submitted to a batch system in place of actual batch jobs. Pilots only acquire resources, but do not perform computational work by themselves. Instead, pilots contact external global batch job schedulers, which push in actual batch jobs as so-called payloads to be executed by the pilot. Pilots are used by a VO to create overlay batch systems that incorporate resources from multiple batch systems. xi, 2, 3, 11, 12, 18, 19, 21, 23, 26, 29–31, 34–38, 109, 131, 149, 158–160, 182

proc filesystem A virtual file system that exposes kernel data and APIs with a directory- and file-like interface. The procfs provides information about processes, but also network, block devices, and other system resources. It is available in the Linux operating system as well as many Unix-like operating systems. 33, 185

Service Availability Monitoring Testing framework and infrastructure validating the availability of services and sites in the WLCG. SAM tests are probes that check both availability and functionality of services and sites by replicating common workflows. Results of these tests are used to rate the availability of sites. 18, 185

simple tree event stream representation Extension of the tree event stream representation. The simple tree event stream representation only supports events referring to the creation of vertices. 53, 54, 57, 71, 72, 108

Subtree-weighted Tree Edit Distance A variant of the TED taking into account the subtree of vertices. While the TED weights all vertices equally, the Subtree-weighted Tree Edit Distance weights each vertex by the number of its descendants. This reflects a hierarchical weight of vertices if children are assumed to be defined by their parent. 133, 135, 186

Subtree-weighted Tree Edit Distance with Moves An extension to the STED differentiating between permutations and moves across branches. Permutations, that is moves between siblings, are not weighted by subtree size, while moves across branches are. This reflects that the path from each subtree vertex to the root vertex is not affected by a permutation of a parent. 134, 135, 186

tree event stream representation Representation of a tree as a sequence of vertex edit operations, such as vertex creation events, vertex deletion events as well as change events. The representation implements several constraints such as the preservation of ancestry of vertices to maintain a valid tree at every point in time. 52–54, 57, 72, 73, 78, 81, 116, 126, 181, 183

Virtual Organisation A Virtual Organisation represents a group in the WLCG. VOs may represent actual organisations, such as the experiments related to the LHC, but also abstract groups, such as administrators of the WLCG sites and services. xiii, 14, 15, 186

XRootD Remote data access protocol and service. The XRootD protocol allows reading of data from local and remote data providers. 16

Acronyms

ALICE A Large Ion Collider Experiment 15, 16

API Application Programming Interface 16, 28, 33, 139

ASSESS Algorithm Simulation for Streaming Environments to aSsess tree Similarities 129, 153, 157, 158, 172, 185

ATLAS A Toroidal LHC Apparatus 15, 16, 18

BPNetMon Batch Process Network Monitor xi, 32, 34, 35, 130, 154, 158, 185

CE computing element 9, 11, 15, 18

CMS Compact Muon Solenoid xi, xiii, 8–12, 14–16, 18, 36, 121, 130, 131, 140, 143, 148, 159, 160, 175, 178

CR Cluster Representative 119–121, 123, 127, 146, 147, 150, 156, 185

DAG Directed Acyclic Graph 59

DTW Dynamic Time Warping 79

FTED Fanout-weighted Tree Edit Distance 132, 133, 135, 136

GMM Gaussian Mixture Model 89, 90, 92, 95, 96

HEP High Energy Physics xi, 1–4, 7, 10, 12, 13, 15, 19–23, 26, 28–32, 35, 37, 38, 66, 67, 114, 129, 133, 137, 139, 144, 147–149, 151, 175

HLT High-Level Trigger 8

HPC High Performance Computing 13, 26

LHC Large Hadron Collider 7–10, 12–15, 183

LHCb Large Hadron Collider beauty 15, 16

MONARC Models of Networked Analysis at Regional Centres for LHC Experiments 8, 9

PDF probability density function 87–89, 91, 92

procfs proc filesystem 33, 185

SAM Service Availability Monitoring 18, 185

SAX Symbolic Aggregate approXimation 79, 80, 93

SE storage element 9, 16, 19, 20

SLA service level agreement 13, 18

STED Subtree-weighted Tree Edit Distance 133–136, 138, 186

STEDWM Subtree-weighted Tree Edit Distance with Moves 134, 135, 186

TED Tree Edit Distance xi, 1, 39–41, 43, 46, 61, 67, 68, 75, 78, 108, 113, 129, 131, 132, 135, 136, 163

UGE Univa® Grid Engine® 16, 157

VO Virtual Organisation xiii, 14–18, 20, 21, 24, 29, 130, 139, 146, 159, 182, 186

WLCG Worldwide LHC Computing Grid xi, 2, 3, 8–10, 13–15, 18, 20, 30, 38, 183