Analyzing Scientific Workflow Management Systems

Bachelor's Thesis of

Daniel Scheerer

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr-Ing. Anne Koziolek
Second examiner:	Prof. Dr. Ralf Reussner
First advisor:	M.Sc. Larissa Schmid
Second advisor:	M.Sc. Timur Sağlam

15. January 2024 – 15. May 2024

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 15.05.2024

. . . . . .

(Daniel Scheerer)

Abstract

Over the last 20 years, scientific workflows have emerged as an important aspect of modern sciences. The abstraction provided by workflows has become a regular aid to handle the high complexity of simulation and computations in many scientific domains. To deal with ever-rising amounts of data and challenges posed by new technologies, scientific workflow management systems have become a valuable tool to orchestrate and monitor the execution of workflows on distributed execution environments. An abundance of research presents new systems, features for those systems, or aims to create synthetic workflows for benchmarking purposes. However, little research focuses on the performance differences of workflow applications when executed with different workflow management systems. This thesis aims to provide benchmarks for multiple workflows and workflow management systems in order to help domain scientists make an informed choice about what system to use. Our measurements show that different workflow management systems do not significantly impact the execution time of workflow tasks.

Zusammenfassung

Über die letzten 20 Jahre haben sich wissenschaftliche Workflows zu einem wichtigen Aspekt moderner Forschung entwickelt. Die von Workflows gebotene Abstraktion ist zu einer regelmäßigen Hilfe im Umgang mit der hohen Komplexität von Simulationen und Berechnungen in vielen wissenschaftlichen Bereichen geworden. Um die immer stärker ansteigende Menge an Daten und die Herausforderungen neuer Technologien zu bewältigen, sind wissenschaftliche Workflow-Managment-Systeme ein wertvolles Werkzeug zur Orchestrierung und Überwachung von Workflows auf verteilten Rechensystemen geworden. Ein großer Teil neuer Forschung beschäftigt sich mit neuen Systemen und ihren Features. Andere verwandte Arbeiten beschäftigen sich mit der Erstellung von synthetischen Workflows für Benchmarking-Zwecke. Jedoch gibt es nur wenig Forschung, die sich mit dem möglicherweise unterschiedlichen Performance-Verhalten von Workflow-Anwendungen beschäftigen, wenn diese mit unterschiedlichen Workflow-Management-Systemen ausgeführt werden. Diese Arbeit zielt darauf ab, Benchmarks für mehrere Workflows und Workflow-Management-Systeme bereitzustellen. Diese ermöglichen es Wissenschaftlern dann idealerweise, eine bessere Entscheidung zu treffen, wenn es um die Wahl eines Workflow-Systems geht. Unsere Messungen ergeben, dass unterschiedliche Workflow-Managment-Systeme keinen signifikanten Einfluss auf die Ausführungszeit von Workflow-Anwendungen haben.

Contents

List of Figures

List of Tables

1. Introduction

In modern times, data science, computation and simulation have become an integral aspect of science besides the established methods of theory and experiment [13]. In this new *e-Science*, software no longer exists exclusively in the form of large monolithic applications [9]. Instead, results are computed by running a complex series of independent tasks on appropriate input data. With this approach, the ability to trace provenance data of results and repeat calculations to reproduce results is an important aspect of the scientific method. To simplify and document this process, the use of *scientific workflows* has been established as an efficient way to model complex processes in an abstract and reproducible way [16]. In recent years, many scientific workflows have analyzed large amounts of data and require ever higher computing capacity [23]. Consequently, most scientific workflows rely on high-performance computing systems like computer clusters, grid networks [15] or high-performance cloud computing. Such systems often provide a heterogeneous computing environment, which necessitates more focus on resource management and workflow orchestration, especially for highly parallel workflows. With this trend, the use of so-called *workflow management systems* (WMS) has become mandatory in order to compose and manage scientific workflows [10, 19]. But since specific requirements concerning the features of such systems differ for each scientific domain, many domains have created their own WMS. This has lead to a plethora of available scientific workflow management systems [29], each with their own format to represent workflows but generally overlapping features. However, it is not clear that all projects from the same scientific domain present the same challenges to a WMS. Additionally, there is a lack of existing benchmarks for different types of workflows to compare the performance of different WMSs. For a scientist who is composing a new workflow to deal with a new set of problems, the best choice of management system regarding performance is not obvious.

This thesis aims to provide benchmarks for a number of selected workflow management systems and workflows. Comparable benchmarking data allows further insight into different classes of workflows and which WMS they work best with. Ideally, this knowledge can help researchers pick a suitable workflow management system for their individual use case. However, since workflow management systems offer different sets of features, performance is not the only quality of WMSs to consider. Concretely, this thesis investigates the following research questions:

RQ1: How can scientific workflows be characterized regarding resource requirements and resource utilization?

RQ2: How do different scientific workflow management systems compare in regard to performance?

RQ3: Is there a class of workflows for which a given management system achieves better performance results than others?

RQ4: What other features, besides performance, have to be considered when selecting a scientific workflow management system?

This thesis is organized as follows: Chapter 2 of this paper explains the foundations of *scientific workflows* and *scientific workflow management systems* in more detail. Chapter 3 details the approach we take for measuring performance metrics and the workflows and systems we choose to analyze. Chapter 4 presents the related work and how this thesis discerns itself from it. In chapter 5, we describe the features and characteristics of the workflow management systems we evaluate. Chapter 6 lists the workflows we characterize along with their structure. All benchmarking data we collect is depicted in chapter 7. The same chapter also details the input instances we use and the limitations of our approach. Chapter 8 discusses the results of our performance evaluation and the different features of WMSs. Finally, chapter 9 summarizes our results and gives an outlook for future work.

The dataset containing the input data and results generated during the work on this thesis are available on Zenodo [27].

2. Foundations

The following chapter gives an overview of the foundations of the central entities in the thesis: Section 2.1 explains the concept of *scientific workflows*. Section 2.2 describes the purpose of *scientific workflow management systems*.

2.1. Scientific Workflows

A *scientific workflow* in its most general form is an abstract representation of a scientific data processing routine [14]. Large scientific simulations or computations are often executed in heterogeneous computation environments. In this case, managing the data transfers to and from the execution sites becomes a complex process [11]. The abstraction provided by scientific workflows allows domain scientists to concentrate on their research instead of the computation management [13, 14].

A workflow consists of tasks (also called jobs) and the data dependencies between them [13]. A task involves the execution of a binary file or shell script to perform computations, data transfers or other auxiliary functions. Data dependencies are usually represented on the file level: Each task takes input in the form of one or multiple files and produces one or multiple output files after it has finished. If a task must wait for the output file of a previous task, they are connected by a data dependency, which implies a temporal ordering. The most widely used representations of workflows are *directed acyclic graphs* (DAGs), but within this framework multiple formats exist. Each task is represented by a node and data dependencies are represented by directed edges between tasks. Figure 2.1 depicts a basic DAG representation. In some formats, files are also depicted as nodes, while edges imply that the file is produced or consumed by a task. Representations other than DAGs exist but are not widely used [13].

Scientific workflows are often further classified into *abstract workflows* and *concrete workflows* [9]. An abstract workflow, detailed in subsection 2.1.1, is independent from a concrete execution environment. Subsection 2.1.2 introduces the concept of concrete workflows, which contain mappings to a specific environment, binaries and input files.

2.1.1. Abstract Workflows

An abstract workflow definition abstracts from an execution environment and the physical locations of the used binaries and input files [9]. For that purpose, all files are referred

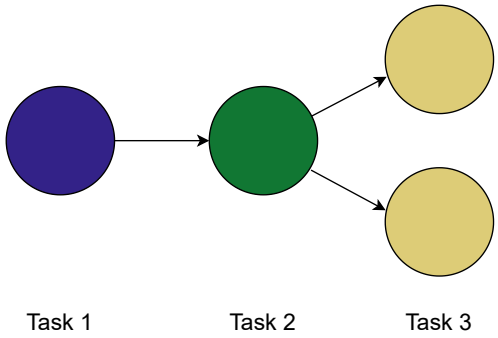


Figure 2.1.: Workflows tasks with data dependencies

to by a logical filename or path relative to the workflow execution directory. Logical filenames offer an even higher level of abstraction than relative paths because they allow to change the structure of the data directory without modifying the workflow definition [9]. Consequently, the DAG of an abstract workflow contains mainly computational tasks decoupled from any specific execution environment. Figure 2.2 depicts the abstract graph representation of an example workflow. This workflow features a classic diamond structure: An input file is scattered, processed in chunks and the results finally merged together. Since it is an abstract workflow, there is no notion of where the tasks are executed and how the data is transferred there. One of the largest advantages of wrapping a procedure in an abstract workflow is the ability to reuse it and reproduce the results on a different infrastructure. This allows scientists to easily verify their results, while sharing them with a community for discussion.

Abstract workflows can also feature additional control structures like conditional branches or loops. In most cases, those structures are just a syntactical convenience, and the values defining the conditional execution must be known while mapping to a concrete workflow. Subsection 2.1.3 goes into more detail concerning workflows that have to adapt their structure during runtime. Another important construct are sub-workflows: these are complete workflows that can be embedded as a part of a larger workflow. This design leads to a more compact workflow definition while not reducing the complexity.

2.1.2. Concrete Workflows

In order to be executed, a workflow must contain specific information about the execution infrastructure, data transfers and physical file locations involved. A workflow representation which includes this information is called a *concrete workflow* [9]. Since including this information nullifies the aforementioned advantages of abstraction, a concrete workflow is often created only directly ahead of the workflow execution. Depending on how distributed an execution system is, input data can be accessed directly through a shared file system or must be transferred to a local storage location first. The same holds true for output data. Intermediate data does not have to be staged out of the execution site if

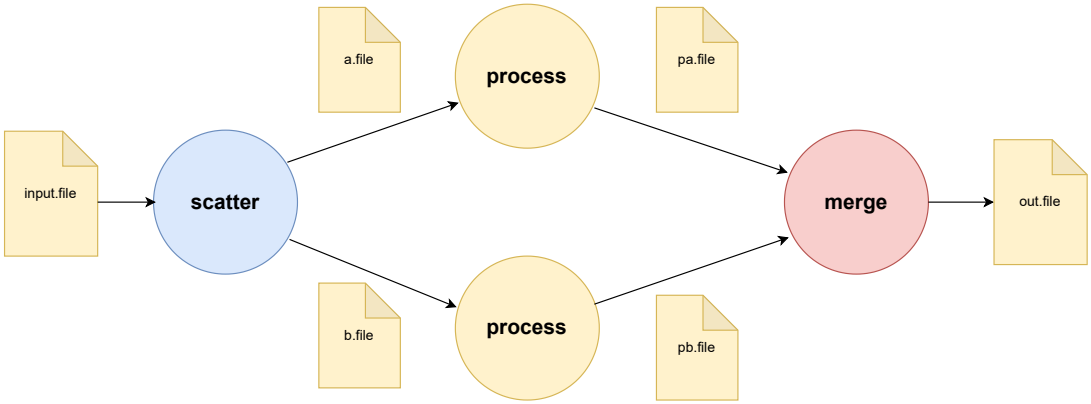


Figure 2.2.: An example of an abstract workflow

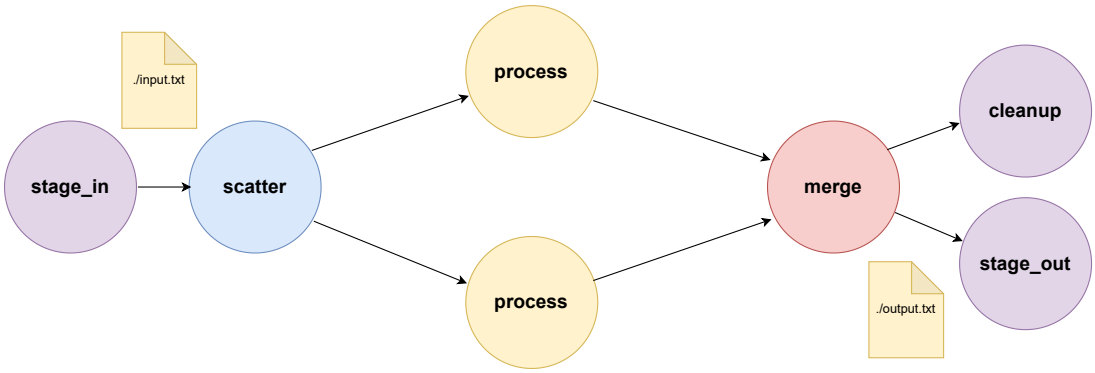


Figure 2.3.: An example of a concrete workflow

it is not an important part of the results. Figure 2.3 depicts a concrete workflow, which includes additional auxiliary jobs to transfer data to and from a remote execution site.

2.1.3. Advanced Characteristics

The demands for what can be represented with a workflow definition are increasing due to rising scale of computations and new technologies [29, 14]. When selecting a WMS, support for certain workflow requirements can play a more important role than pure performance. While the workflows we profile in this thesis do not display the following characteristics, they are important to know about when talking about the features of WMSs.

Adaptive Workflows In some cases, workflows need to be adjusted during their runtime according to the results of their jobs. An example of such workflows are those that utilize machine learning to improve simulation parameters [29]: The workflow structure must be modified during execution to fit a new configuration. This confronts workflow

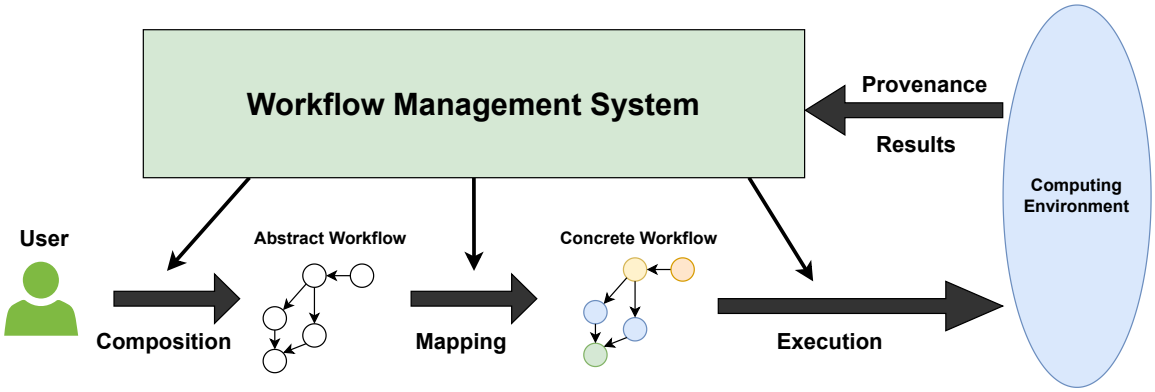


Figure 2.4.: The core features of a workflow management system

management systems with a new challenge since the complete workflow structure is not known while mapping the workflow to the resources of an execution site.

In Situ Workflows In situ workflows are workflows defined by tightly coupled tasks which exchange information [12] during their execution. Especially in the context of extreme-scale computing, the volume of simulation data that has to be analyzed can become too large for the storage bandwidth [14]. In this case, integrating the data analysis within the simulation on the compute site is a necessary step. However, the communication of tightly coupled tasks is often done via memory [14]. In contrast to this, most WMSs only support data transfers through files.

2.2. Scientific Workflow Management Systems

A *scientific workflow management system* (WMS) is a piece of software designed to help with the creation and execution of scientific workflows. Especially for distributed and heterogeneous execution sites, a WMS offers abstraction and automation features that free an user from considering the requirements of a specific site or execution engine. Subsection 2.2.1 lists the core features which can be found in most workflow management systems, while subsection 2.2.2 introduces additional features that have proven to be useful for researchers.

2.2.1. Core Features

Figure 2.4 depicts the central features of workflow management systems. These core features were structured in more detail in previous work [13]:

Workflow Composition The WMS helps with workflow creation by providing either a graphical or textual interface [4]. Graphical interfaces help researchers without programming experience define and visualize their workflow by directly modifying the workflow graph [4]. Textual interfaces work through either a higher-level scripting language or a command line interface. This approach offers higher scalability and more control over complex structures than graphical interfaces, for example if the problem has to be split up into multiple sub-workflows [14, 13].

Workflow Mapping is the process of assigning specific physical computing resources to the different tasks of a workflow, essentially turning an abstract workflow into a concrete one. This can be done statically or dynamically during runtime. The output of this stage can range from complex files, which can only be executed by a specific engine or scheduler to simple batch scripts. If the abstract workflow definition references files by logical file names, this step is responsible for associating them with a real physical file and path. If the WMS has access to multiple different execution sites, it has to choose an appropriate site for each task. In any case, the abstract workflow tasks must be wrapped in a format compatible with the available sites.

Workflow Execution refers to the actual processing of the mapped workflow. Large or distributed execution environments make use of so-called *batch systems*. These systems work as execution engines and are responsible for monitoring the available compute resources and scheduling user requests for computing power in a fair and efficient manner [2]. However, the interfaces for scheduling jobs, checking their status and collecting the outputs differ for each execution engine. WMSs usually offer support for a variety of execution engines and environments. During execution, they are responsible for submitting workflow jobs in a correct manner and periodically checking their status to resume jobs that depend on their output.

Provenance Recording is the task of collecting additional metadata about the workflow and the results. This involves checking the return values of workflow jobs for errors and saving the log files of the respective execution engine. This practice is important to verify research results and increase their reproducibility.

In practice, most WMSs were created by scientific communities from a certain domain and influenced by the specific needs of that domain. Thus, the features of WMSs apart from core features can differ.

Presently, WMSs have to deal with a new set of challenges arising from advanced technology [29]. The development of exascale computing makes efficient management and coordination between compute resources, especially for parallel applications, more important than ever. New AI-aided workflows require more flexible and dynamic systems, which are able to react to changing parameters and adapt the structure of the workflow during execution.

2.2.2. Other Features

Apart from the set of core features that most management systems support in some way, WMSs can offer a multitude of additional features. These often allow for more robust or versatile workflow execution and should be considered when choosing a WMS.

Active monitoring Assuming that a workflow is not executed on the local machine, which is also running the engine of a WMS, management systems have no direct control over the individual jobs that have been submitted to an execution engine. Some systems can prevent this by submitting a pool of worker threads to the batch system and communicating with these workers directly over a network connection. This approach allows for finer control over the execution on site. Another benefit is the reduced overhead from the batch system, since new jobs do not have to be submitted separately [2]. This is especially helpful for workflows with thousands of short jobs with an execution time below a minute.

Fault Tolerance Especially for workflows with hundreds of thousands of jobs and heterogeneous execution environments, failures during execution have to be expected [2]. While all systems can detect a failed job within a workflow execution, not all of them are able to automatically recover from such an occurrence. The most basic tool for recovering is to simply restart any failed jobs. If available, restarting it in a different execution environment can be helpful to prevent the failure. Other means include keeping a logging file of the execution to resume a workflow even after the execution engine of the management system was shut down or crashed [2].

3. Approach

The primary research goal of this thesis is to offer comparable performance benchmarks for scientific workflows and workflow management systems. For this purpose, we characterize three scientific workflows from different domains and analyze the performance of three workflow management systems with each workflow.

The characterization segment in chapter 7 focuses on the resource requirements and resource utilization of the respective workflow. To understand the scaling of these characteristics for varying input sizes, we use 5 different problem instances for each workflow. We choose this number according to related work [6] in order to allow modeling of performance behavior depending on input parameters. We execute each workflow instance on a single compute node of the cluster without the help of a workflow management system. By running the workflow tasks in parallel on the same node, we analyze the negative performance impact of having multiple task instances share the same main memory and CPU. This allows us to evaluate the potential performance gain by submitting each job individually in contrast to an increase in submission overhead.

We compare workflow management systems by executing each workflow instance with each system. For this, we create a distinct workflow definition according to the demands of each WMS. We then compare the runtime for the different systems per task.

3.1. Performance Metrics

In order to evaluate the performance of software, multiple metrics can be observed. Characterizations in previous work have measured time [19, 2, 7], CPU utilization [19], peak memory usage [19], and I/O throughput [19, 2, 21, 7]. In this work, we focus on the following performance metrics when characterizing workflows and comparing the different management systems to each other:

Execution Time is the most practical performance metric, since a fast execution is the desired outcome of any performance optimization process. Measuring the elapsed time during a workflow run allows the results to be compared to other workflow management systems.

Floating Point Operations per Second (FLOPS) are the primary measure of computing efficiency. The theoretical maximum of FLOPS for a given CPU depends on the clock frequency and architecture. Whenever a process has to wait, for example, for slower I/O operations to complete, the number of FLOPS falls below this optimum. Consequently, FLOPS are a useful performance metric to measure how well a workflow utilizes the given CPU time.

Load/Store Instructions per Second (LSPS) are a measure of data transfer load. A high number of load/store instructions together with a lower than average number of FLOPS indicates an I/O intensive task.

Main Memory Usage denotes the peak RAM usage of a process. Although the memory usage of a workflow is hard to improve for a management system, it is still a relevant characteristic for improving the overall wall clock time of a workflow run. If multiple workflow jobs require the same data, grouping and executing them on nodes with shared memory or caches can reduce the I/O overhead. The higher the memory usage of a job, the more impactful the reduction of this overhead can be.

3.2. Selection of WMS

In this work, we focus on three established workflow management systems with different design philosophies. Since we execute all workflow instances on a high-performance cluster with the Slurm batch system, support for this specific computing environment and execution engine is a necessary feature. We try to select systems that have relevance in actual research, have been used for comparison or characterization attempts before, and come from different scientific domains.

Pegasus [10] is one of the most commonly used scientific workflow management systems. For example, the execution of the famous *LIGO* [1] workflow that detected gravitational waves was performed with the *Pegasus* system. It has been the subject of multiple papers analyzing its characteristics [14, 32, 22] and was used for the purpose of profiling workflows before [19].

Makeflow [2] is a workflow system with a focus on large distributed systems and data-intensive parallel workflows. The workflow definition syntax used is similar to Make. *Makeflow* has been primarily used in the physics domain and has been the subject of previous characterizations [14].

Snakemake [20] is a Python-based workflow engine from the bioinformatics domain, which composes workflows via its own definition language. It can run on all kinds of computing environments without modifying the workflow. It also hosts a repository of public workflows that fulfill the *Snakemake* standards [31].

4. Related Work

This chapter presents the scientific work related to the topic of this thesis. A number of related works deals with the characterization and profiling of workflows and workflow management systems. Section 4.1 introduces the work which has addressed the characterization of scientific workflows. Section 4.2 presents the work that has concerned itself with the comparison and analysis of scientific workflow management systems. For each work, we will detail what sets our contribution apart from the other approach.

4.1. Workflows

This section deals with the related work, which focuses on the characterization and benchmarking of scientific workflows.

Albrecht et al. [2] present a suite of workflow benchmarks with the name *workbench*. These benchmarks are small synthetic workflow definitions that aim to cover different possible workflow structures and characteristics. The characteristics include dispatch overhead of the execution engine, job throughput, I/O throughput, and interprocess communication. Each workbench workflow isolates and focuses on one of these characteristics. They argue that workflows with a long runtime and few I/O operations are unfit to determine the performance. Albrecht et al. then evaluate the benchmarks for two different execution site architectures and with four different execution engines. However, they only use their own *Makeflow* workflow management system for execution, which they also present in their paper. Their results show that the dispatch overhead of the execution system determines the upper limit for job throughput. Local execution shows a very low dispatch latency of under 0.05 seconds, while batch systems like *HTCondor* [18] and *Hadoop* show high latency values up to 30 seconds. They also show that their own WorkQueue execution engine has low latency times of under 0.1 seconds, close to local execution. The WorkQueue system can be seen as an active monitoring tool, as described in subsection 2.2.2. Our work discerns itself from this approach by analyzing and comparing multiple workflow management systems, not just one. We also focus more on the performance of the isolated workflow tasks themselves rather than the job throughput on a workflow level. Another difference is our use of real scientific workflows, while workbench only offers synthetic workflows.

Coleman et al. [8] present a framework for the creation of synthetic workflows modeled after real examples that are given as input. They introduce a set of tools that can analyze the characteristics of a given workflow execution log and extract relevant parameters. This is used as input to a generator which can then create synthetic instances of the real workflow.

Ramakrishnan et al. [26] present and characterize a number of workflows from the meteorology, bioinformatics, physics and computer science domains. Their profiling data is limited to the execution time for each subtask and the file sizes passed along the data dependencies. They discuss characteristics like degree of parallelism or number of I/O operations. However, the work does not include the influence of the WMSs in its analysis. Instead, workflows were executed with varying systems like Taverna or Kepler, which makes a comparison impossible. In contrast, our work executes each workflow with each WMS.

Krol et al. [21] created performance profiles for a single example workflow. This work focuses on the distribution of resource requirements over time, not only the peak requirement. The executions were performed with the *Pegasus* workflow management system [10].

Juve et al. [19] presented and characterized six workflows from various scientific domains. The focus of this work was to create task-level performance profiles of the workflows by using novel profiling tools that capture data about runtime, memory usage, CPU utilization and I/O workload. However, the profiling was performed using only the *Pegasus* workflow management system and different execution environments for each workflow.

4.2. Workflow Management Systems

Much related work in regard to the characterization of WMSs focuses on features other than performance. WMSs have been characterized in regard to their conditional workflow capabilities [3], parallelization and scheduling techniques [4, 23], or general workflow representations [32].

Da Silva et al. [14] characterized and compared workflow management systems in regard to their extreme-scale computing capabilities. This work included the comparison of relevant attributes like workflow execution models, capabilities for heterogeneous computing environments, and the data access methods supported by the compared systems.

In summary, our work contrasts itself by the following points: We use only a singular execution engine, Slurm, and a single execution site, namely a high-performance cluster, to evaluate the performance. However, we analyze multiple real workflows instead of synthetic ones and observe and compare the performance of multiple workflow management systems.

5. Workflow Management Systems

The following chapter contains an overview of the workflow management systems that we analyze in more detail in this thesis. Each section gives detailed information about the options for workflow composition and execution for the respective management system. At the end of this chapter, section 5.4 gives a summary of the important core features and characteristics of each system. For each WMS, we give a short workflow composition example for the diamond structure workflow presented in subsection 2.1.1.

5.1. Pegasus

Pegasus [10] is an open-source workflow management system for composing, mapping and executing scientific workflows on different computational infrastructures [10]. It has been used in multiple scientific domains like astronomy, seismology, bioinformatics and physics [10]. The design of *Pegasus* puts a strong emphasis on differentiating abstract from concrete workflow definitions, which leads to improved scalability and flexibility.

5.1.1. Workflow Composition

Pegasus offers no graphical user interface for workflow composition. Abstract workflows are defined in a textual fashion in the YAML format. Since directly writing a definition in YAML is complicated, *Pegasus* instead offers programming APIs for the Python, Java and R languages. *Pegasus* strictly discerns abstract workflow definitions from concrete workflow mappings, as explained in subsection 2.1.2. Abstract definitions are universal and can be executed on any environment, like local computers, high-performance clusters, grids or cloud services. To achieve this separation, abstract workflows do not associate executable and input files with file system paths. Instead, they are only referenced by a logical filename. This also means that the abstract definition does not change when the storage layout of data changes.

The following code shows the workflow definition for the example workflow from Figure 2.2 with the *Pegasus* Python API:

```
wf = Workflow("Example")

input_file = File("input.file")
out_file = File("out.file")
filenames = ["a", "b"]
```

```
6
7  scatter_job = Job("scatter")
8  scatter_job.add_inputs(input_file)
9  scattered_files = []
10 for filename in filenames:
11     output = File(filename + ".file")
12     scattered_files.append(output)
13     scatter_job.add_outputs(output)
14
15 scatter_job.add_args(input_file)
16 wf.add_jobs(scatter_job)
17
18 processed_files = []
19 for input in scattered_files:
20     process_job = Job("process")
21     process_job.add_inputs(input)
22     processed_file = File("p" + input.lfn)
23     processed_files.append(processed_file)
24     process_job.add_outputs(processed_file)
25     process_job.add_args(input)
26     wf.add_jobs(process_job)
27
28 merge_job = Job("merge")
29 for input in processed_files:
30     merge_job.add_inputs(input)
31     merge_job.add_args(input)
32 merge_job.add_outputs(out_file)
33 wf.add_jobs(merge_job)
```

Listing 5.1: Pegasus workflow definition

5.1.2. Workflow Execution

Before an abstract *Pegasus* workflow can be executed, it must be transformed into a concrete workflow which includes mappings to compute resources, physical files and even additional data transfer jobs inserted by *Pegasus* itself. This transformation is done by the *Pegasus-plan* tool. Since for some environments the physical locations of files or executables can deviate from the abstract definition, *Pegasus* uses multiple so-called catalogs during the mapping process. The resulting workflow is ready to be submitted, but if the environment changes, the planning must be repeated.

5.2. Snakemake

Snakemake is a workflow management system from the bioinformatics domain [20]. It requires a Python installation to work. *Snakemake* offers no graphical user interface. Instead, workflow composition is performed with a domain-specific language in a textual

way in so-called *Snakefiles*. *Snakemake* workflows can be used in different execution environments without changing their definition.

5.2.1. Workflow Composition

The syntax of the domain-specific *Snakemake* language is similar to the Python language, and the design philosophy of workflows is based on the build system *make*. Workflows are defined by a set of rules with corresponding input and output files. Each rule provides a shell command or Python script to generate its output files. A complete workflow requires the definition of target files or rules. Those files, or the output files of the target rules respectively, are the desired final output of the workflow. When building the concrete workflow, *Snakemake* works with a backwards approach starting from the target output. If any output file does not already exist in the file system, *Snakemake* searches for a rule capable of providing this output. If the input files of this next rule do not exist, the process is repeated recursively. This approach leads to a *directed acyclic graph* defining the dependencies and inputs of all rules necessary to create the final output, which implicitly defines the complete workflow.

The following code shows the workflow definition for the example workflow from Figure 2.2 in Snakemake specific syntax:

```
FILENAMES = ["a", "b"]
rule scatter:
    input:
        "input.file"
    output:
        expand("{id}.file", id=FILENAMES)
    shell:
        "scatter -i {input} -o {output}"

rule process:
    input:
        "{id}.file"
    output:
        "p{id}.file"
    shell:
        "process -i {input} -o {output}"

rule merge:
    input:
        expand("p{id}.file", id=FILENAMES)
    output:
        "out.file"
    shell:
        "merge -i {input} -o {output}"
```

Listing 5.2: Snakemake workflow definition

The merge and scatter jobs use expand-syntax to allow varying sizes of intermediate jobs. The process rule utilizes a wildcard to offer dynamic behavior.

The above code can be executed with the following command, specifying the desired target output:

```
snakemake out.file
```

A noteworthy aspect of *Snakemake* is that "it is the first system to support the use of automatically inferred multiple named wildcards (or variables) in input and output filenames" [20]. The benefit of this feature is that a single rule in the abstract workflow can be used to create multiple jobs in the concrete workflow. Consequently, the workflow definition in the Snakefile can stay the same while executing input instances of varying size.

5.2.2. Workflow Execution

Snakemake performs the workflow execution directly in the directory of the Snakefile. It requires all jobs to have direct access to a shared file system. Thus, it is not compatible with distributed execution environments like grids. This restriction allows *Snakemake* to perform without dedicated stage-in or stage-out jobs of intermediate data. It does support local and cluster execution with different batch systems. When evoking *Snakemake*, the user needs to specify a target file or rule. The *Snakemake* engine then constructs the workflow execution and dependency graph in a backwards fashion: If the input files for the target rule do not exist, *Snakemake* searches for rules capable of producing those files. This pattern is applied recursively until every required input is already available. If a required file does not exist and there is no rule to produce it, the workflow planning fails. After successful planning, the workflow is immediately executed. Additionally, *Snakemake* is able to perform dry-runs to build and display the workflow plan without executing it. It is also possible to print out the workflow DAG in the dot format to create a graphical representation.

5.3. Makeflow

Makeflow [2] is an open-source workflow engine which is part of the *Cooperative Computing Tools* software package developed by the *Cooperative Computing Lab* at the University of Notre Dame. It is designed for large-scale distributed computing on local clusters or remote machines. *Makeflow* is used in multiple scientific domains, like bioinformatics [2] or high-energy physics [14].

5.3.1. Workflow Composition

Makeflow offers no graphical user interface for workflow composition. Instead, it offers two different textual approaches: The first is a custom language inspired by classic make syntax. The second is the JX workflow language, which is an extended form of JSON.

Classic make style For this approach, the workflow definition consists of rules and assignments [2] with a syntax very similar to classic make. Each rule specifies how to create a single or multiple intermediate target files that are part of the workflow. For this, the rule names all required input files and a shell command that specifies how to create the target file. *Makeflow* assumes that if all input files are present on the local file system, the target rule can be executed. Since *Makeflow* also supports execution sites without shared file systems, in this case the syntax of rules differs from classic make [2]: The rules must declare all input files as dependencies without exception. This allows the *Makeflow* execution engine to copy required data to remote sites if necessary. By using the *LOCAL* keyword, users can instruct *Makeflow* to execute individual tasks locally [2]. While simple, the make style workflow definitions offer no abstract concepts like jobs, files or wildcards. As a consequence, a user has to create a "hard-coded" definition, and for each job in the final workflow, a matching rule is required. For example, the *process* task from Figure 2.2 in chapter 2 requires a new rule for each execution. The number of parallel *process* tasks cannot be extended without modifying the definition. In practice, the user has to write a custom generation script which generates the make file. In this paper, we use this option for defining workflows.

JX workflow language This definition approach describes the workflow with a single JSON object. In this object, rules are defined as a list of rule objects stored under the "rules" key. Rule objects can represent single commands or complete sub-workflows. The JX language can be combined with the JX expression language extension. This allows for more flexibility when defining workflows.

The following code shows the workflow definition for the example workflow from Figure 2.2 in *Makeflow* make syntax:

```
out.file: pa.file pb.file
    merge pa.file pb.file

pa.file: a.file
    process a.file

pb.file: b.file
    process b.file

a.file b.file: input.file
    scatter input.file
```

Listing 5.3: Makeflow workflow definition

5.3.2. Workflow Mapping and Execution

Once a *Makeflow* workflow is composed, it can be executed in different computing environments without adjusting the workflow definition. *Makeflow* combines the mapping and execution phase in the *Makeflow* command. Execution is supported for local machines,

WMS	Pegasus	Snakemake	Makeflow
Composition	Python-API	domain-specific language	make
Wildcards	✗	✓	✗
Cluster Support	✓	✓	✓
Grid Support	✓	✗	✓
Cloud Support	✓	✓	✓
Job Grouping	✓	✓	✓
Auxillary Jobs	✓	✗	✗
DAG visualization	✓	✓	✓

Table 5.1.: Comparison of WMS features

high performance clusters with batch systems, grids and cloud services. Through configuration, *Makeflow* can be instructed to exclusively use an execution site or dynamically choose a suitable site for the tasks individually. *Makeflow* keeps track of submitted jobs through the interface of the execution engine in use. During execution, all submissions and their return values are stored in a transaction log. Furthermore, *Makeflow* offers a high level of fault tolerance: Failed jobs are automatically resubmitted and if the whole engine crashes, a *Makeflow* workflow can be picked up again mid-execution after a restart.

For cluster or grid execution, *Makeflow* can be used together with the Taskvine system, another tool in the *Cooperative Computing Tools* software package. In this mode, single jobs are not submitted through the batch system. Instead, a number of worker jobs is submitted and tasks are dispatched to the workers directly over a network connection. This approach allows for job grouping and can significantly reduce the submission overhead of batch systems, which is usually up to 30 seconds per job. Additionally, this can reduce the amount of duplicate data transfers if multiple jobs share the same input data.

5.4. Comparison of Core Features

Table 5.1 compares the core features of the three workflow management systems we choose to examine. We will discuss these features in more detail in chapter 8.

6. Scientific Workflows

This chapter details the scientific workflows that we characterized and used for performance benchmarks in this thesis. For each workflow, the workflow graph structure and different job types are examined.

6.1. Montage

Montage[1] is a toolkit of image processing applications from the astronomy domain. *Montage* is capable of creating high-resolution mosaics of wide areas of the sky, which are too large to be captured by a single telescope image.

Chaining the different tools together results in a workflow structure. The *Montage* workflow takes a set of astronomical *Flexible Image Transport System* (FITS) images with a common frequency band as input and creates a complete custom mosaic image of the sky as output. This output can either be in the FITS format itself or in a common image format like *PNG* or *JPG*. If more than one frequency band is used for the input images, *Montage* can create colored output images by associating each frequency band with a color in the visual spectrum.

Montage was initially released in 2003. Due to its age and open-source nature, *Montage* has been used regularly as a benchmark to assess the performance and structure of scientific workflows.

Figure 6.1 depicts the structure of a *Montage* workflow graph along with its jobs. To begin, each input image must be reprojected to fit the viewing plane of the desired output mosaic. This is done by *mProject* jobs, one for each input image. The projection jobs are independent and can be executed in parallel. Next, *mDiffFit* jobs calculate the differences between each pair of overlapping images. These jobs are again independent for each image pair. A single *mConcatFit* job merges these differences together. The *mBgModel* job uses the merged differences to compute a global background correction, which is then applied to each reprojected image by *mBackground* jobs. Like the initial projection jobs, all mBackground jobs can be run in parallel. The *mImgtbl* job simply constructs a new metadata table for the corrected images. The actual mosaic assembly is performed by the *mAdd* job, which outputs the final mosaic in FITS format. Finally, the *mViewer* job creates a visual representation, in the case of our workflow, a grayscale PNG image.

[1] https://github.com/Caltech-IPAC/Montage

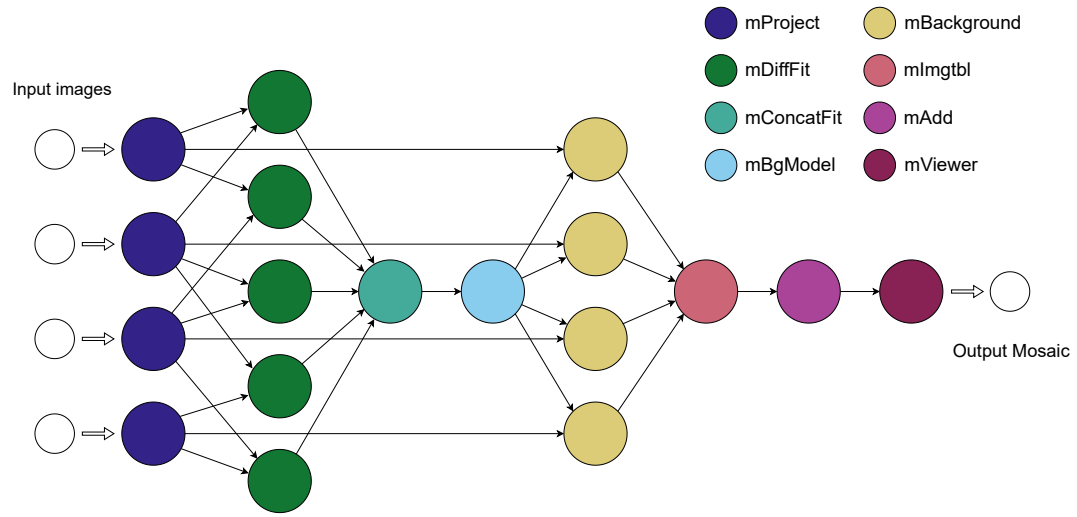


Figure 6.1.: *Montage* workflow structure

6.2. 1000 Genome Workflow

The *1000Genome*[2] workflow is an example workflow from the bioinformatics domain hosted on the official GitHub of the Pegasus-WMS project. This workflow uses data on human variation acquired by the *1000 genomes* project to identify mutational overlaps and allow a statistical evaluation of potentially disease-related mutations. For each chromosome to be examined, the workflow takes a file listing all Single nucleotide polymorphisms (SNPs) variants located in that chromosome and information about which individual has which variant.

Figure 6.2 depicts the structure of a *1000Genome* workflow graph along with its jobs. Each workflow instance has a set number of *individuals* jobs. These jobs parse the chromosome data and a file describing the columns of the data table. If more than one individuals job is used, they each parse only a chunk of the input data. The *individuals_merge* job merges these chunks together. Independent of the individuals jobs, the *sifting* jobs compute the so-called *SIFT score* of all SNPs variants present in a chromosome. The SIFT score of a mutation indicates how harmful it is. Both the SIFT score and the individuals data extracted are then utilized by two different job types: *mutation_overlap* jobs compute the overlap in SNPs variants among pairs of individuals, while *frequency* jobs measure the frequency of mutational overlaps.

[2]https://github.com/pegasus-isi/1000genome-workflow

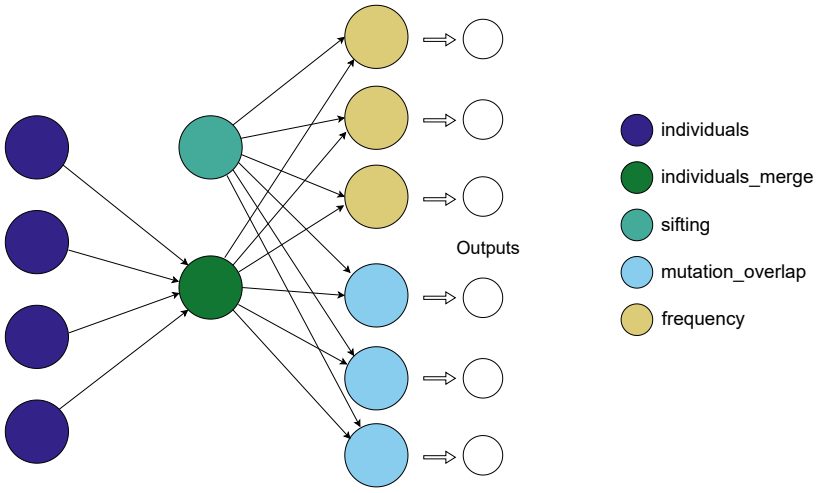


Figure 6.2.: *1000Genome* workflow structure

6.3. Orcasound Workflow

The *Orcasound*[3] workflow is a workflow from the bio acoustics domain hosted on the official GitHub of the Pegasus-WMS. This workflow uses audio stream data provided by the *Orcasound Project* [24], which maintains multiple hydrophones in the pacific northwest to record and monitor southern resident orcas. The *Orcasound* workflow aims to recognize orca sounds in the live-streamed audio with the help of machine learning models.

Figure 6.3 depicts the structure of an *Orcasound* workflow graph along with its jobs. The audio data provided by the Orcasound Project is divided into chunks, which are about six hours long each. Each chunk is identified by a timestamp that represents the start time of the audio. Since the data is taken directly from the live stream, each chunk consists of thousands of small audio files in the *transport stream* format (.ts), stored in a single directory. The *convert2WAV* jobs of the workflow each take a directory of a timestamp as input and convert all transport stream files within into WAV files. *Convert2spectrogram* jobs then create visual representations of the frequency spectrum for each directory of WAV files. Independently, the *inference* jobs use the WAV files and a pre-trained machine learning model to predict the probability that an orca can be heard for each timestamp. The confidence of predictions is stored in a JSON file. Finally, all predictions from all timestamps are merged together into a single JSON file.

[3]https://github.com/pegasus-isi/orcasound-workflow

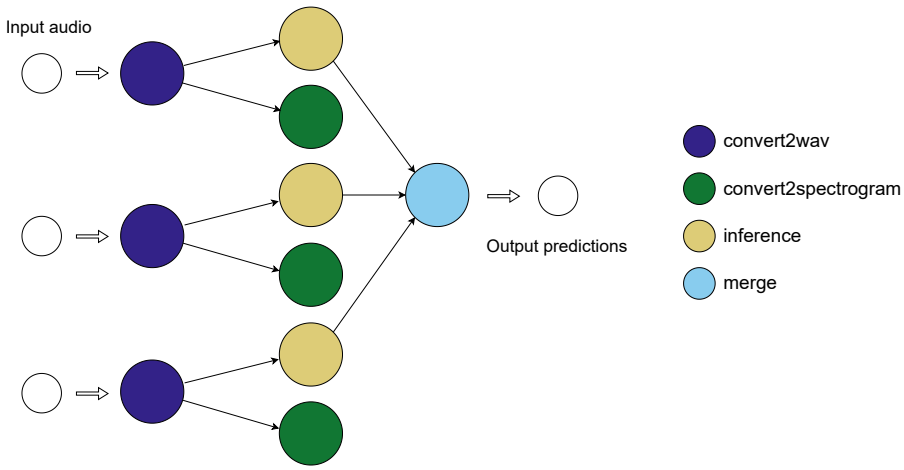


Figure 6.3.: *Orcasound* workflow structure

7. Performance Evaluation

The following chapter presents the collected profiling data. We profile each workflow in more detail without a workflow management system to investigate its inherent characteristics. We then perform performance measurements for each combination of workflow management system and scientific workflow.

To begin with, we detail our methodology when evaluating performance in section 7.1. We then present the performance metrics of our workflow runs, grouped by workflow: section 7.2 shows the results for the *Montage* workflow, section 7.3 the results for the *1000Genome* workflow and finally section 7.4 the results of the *Orcasound* workflow execution runs. At the end of this chapter, section 7.5 examines the threats to validity of our approach.

7.1. Methodology

We perform all computations on the bwUniCluster 2.0 high-performance cluster [5]. All jobs are submitted in the form of batch jobs with the Slurm [30] workload manager (version 23.02.6). To characterize the workflows without a workflow management system, we execute all input instances on a single cluster node with two CPU sockets and 20 cores per socket, for a total of 40 cores per node. All nodes in use are equipped with *Intel Xeon Gold 6230* processors. We use the Score-P [28] measurement infrastructure (version 7.1) and the PAPI [25] interface to collect all profiling data, for both characterization of workflows and comparison of management systems. For workflow tasks in *Python*, we use the Python Score-P bindings [17] (version 4.4.0). With *Score-P*, we use manual instrumentation of only the main method. By not profiling all call paths of the workflow tasks, we reduce the measurement overhead. All performance data in the tables in this section is averaged over five repetitions to eliminate the effects of variation caused by, for example, system noise. We calculate the coefficient of variation (CoV) for this average and note when it has a value greater than 0.1.

During characterization, the different workflows stages are executed sequentially, while parallel jobs within a stage are executed concurrently. For instances where the number of parallel jobs of the same type exceeds the number of cores, we execute at most 40 jobs concurrently and wait for their execution to finish before resuming with the rest.

For workflow runs with workflow management systems, we follow a different execution model. All workflow management systems queue each task as an individual submission in

Instance ID	Square area size in degrees	Input images	Total Jobs
0	0.25	6	28
1	0.50	19	91
2	1.50	99	497
3	2.00	162	830
4	2.50	245	1268

Table 7.1.: Input instances of the *Montage* workflow in this thesis

the Slurm batch system. We configure the systems to always request a complete node with all cores and memory to prevent the side effects of other jobs influencing the measurements. We also limit each workflow management system to only submit a maximum of 25 jobs at a time. This approach increases the dispatch overhead on a workflow level. However, since we work on a shared high-performance cluster with other users, we do not measure the wall clock time of a workflow due to the varying wait times for job submissions to start.

The following sections present the results of our measurements in the form of tables and graphs. Where meaningful, we present a performance model created by *Extra-P* and name the adjusted coefficient of determination (R^2), which indicates how well the model fits the data points.

7.2. Montage

This section presents the performance evaluation of the *Montage* workflow. We compile the *Montage* binaries with the GNU compiler collection (version 11.2).

7.2.1. Input Instances

We profile the *Montage* workflow with five different input instances of varying angular size, but all centered on the astronomical object *Messier 51* (also known as the *Whirlpool Galaxy*). The input images used were taken by the *Sloan Digital Sky Survey* (SDSS) at ultraviolet wavelengths of around 354.3 nm. Since only one frequency band is used as input, the output image is generated in grayscale. The resolution of the output is one pixel per arcsecond.

Table 7.1 shows the individual characteristics of the input instances that were used for the *Montage* workflow. The main parameter influencing the number of images per instance is the size of the observed area in the final mosaic. This parameter is given in degrees and applies to both the horizontal and vertical axes. The number of different jobs corresponds to the number of pairs of overlapping images. Table 7.2 shows the number of all jobs by job type for all instances.

Instance ID	mProject	mDiffFit	mConcatFit	mBgModel	mBackground	mImgtbl	mAdd	mViewer
0	6	11	1	1	6	1	1	1
1	19	48	1	1	19	1	1	1
2	99	294	1	1	99	1	1	1
3	162	501	1	1	162	1	1	1
4	245	773	1	1	245	1	1	1

Table 7.2.: Numbers of jobs by job type for all *Montage* instances

7.2.2. Characterization

This section characterizes the montage workflow regarding resource utilization. All Montage instances are executed on cluster nodes with 96 GB of memory.

Workflow Level Table 7.3 depicts general data of the workflow runs for all five input instances. Note that the walltime values contain an overhead of roughly 30 seconds due to the Slurm batch system. For our input instances, *Montage* has an overall short runtime, which is below 10 minutes, even for the largest instance. However, it features numerous small jobs. The memory usage is low, with a peak usage of 1.4 GB.

Instance ID	Total Jobs	Workflow Walltime (s)	Peak Memory Usage (MB)
0	28	62.2	86.3
1	91	70.2	321.1
2	497	183.4	547.4
3	830	288.6	908.0
4	1268	416.9	1414.0

Table 7.3.: Montage Workflow Level

The following paragraphs and tables examine the *Montage* workflow on a job level by observing the isolated performance metrics of all jobs independently.

mProject Table 7.4 depicts the performance metrics of mProject jobs. These projection jobs represent the largest portion of the workflow walltime. Executing up to 40 jobs in parallel raises the runtime of an individual job by up to 15 seconds, probably due to limiting memory bandwidth. With a CPU usage of around 1 GFLOPS, this job type can be considered compute-intensive.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	199.87	33.31	33.01	33.65	1.07×10^9	3.91×10^9	76	12	12
1	714.48	37.60	34.88	38.59	9.43×10^8	3.53×10^9	76	12	12
2	3739.62	46.75	39.19	49.64	7.60×10^8	2.79×10^9	76	12	12
3	7461.17	46.63	39.75	49.17	7.63×10^8	2.81×10^9	76	12	12
4	11 454.69	47.73	40.33	51.31	7.46×10^8	2.73×10^9	76	12	12

Table 7.4.: *Montage*: **mProject** job data

mDiffFit Table 7.5 depicts the performance metrics of mDiffFit jobs. Each one of these jobs has a short runtime of under 0.5 seconds. They still read more file data than mProject jobs and have high L/SPS values. These jobs can be considered I/O-intensive.

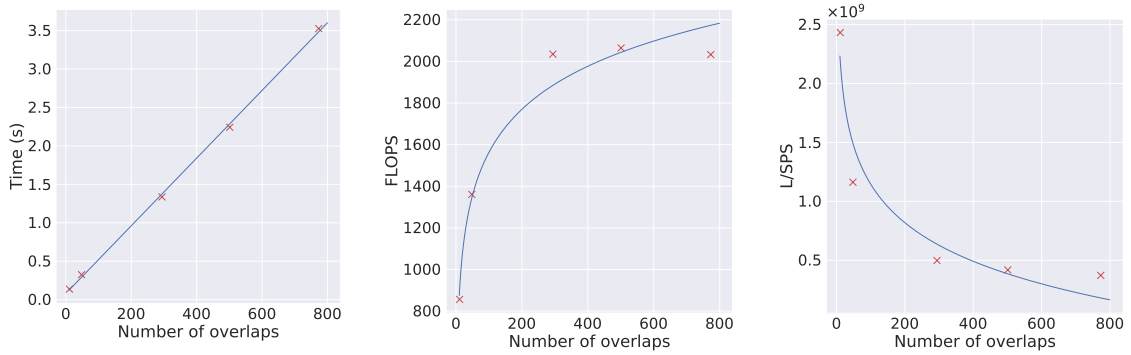
	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	2.42	0.27	0.23	0.31	6.16×10^6	8.04×10^8	4	24	1
1	12.19	0.37	0.25	0.48	4.70×10^6	5.34×10^8	4	24	1
2	85.77	0.36	0.21	0.54	4.84×10^6	5.71×10^8	4	24	1
3	145.27	0.35	0.20	0.56	4.88×10^6	5.83×10^8	4	24	1
4	230.82	0.36	0.20	0.62	4.86×10^6	5.68×10^8	4	24	1

Table 7.5.: *Montage*: **mDiffFit** job data

mConcatFit Table 7.6 depicts the performance metrics of mConcatFit jobs. This is a short job, its runtime scaling up with the number of input images. This job has a low number of FLOPS while maintaining average numbers of Load/Store instructions per second. Consequently, it can be considered I/O-intensive.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	0.14	-	-	-	856.40	2.43×10^9	4	<1	1
1	0.33	-	-	-	1360.81	1.16×10^9	4	<1	1
2	1.34	-	-	-	2035.04	4.98×10^8	4	1	1
3	2.24	-	-	-	2064.64	4.18×10^8	16	2	1
4	3.53	-	-	-	2032.78	3.72×10^8	132	3	1

Table 7.6.: *Montage*: **mConcatFit** job data



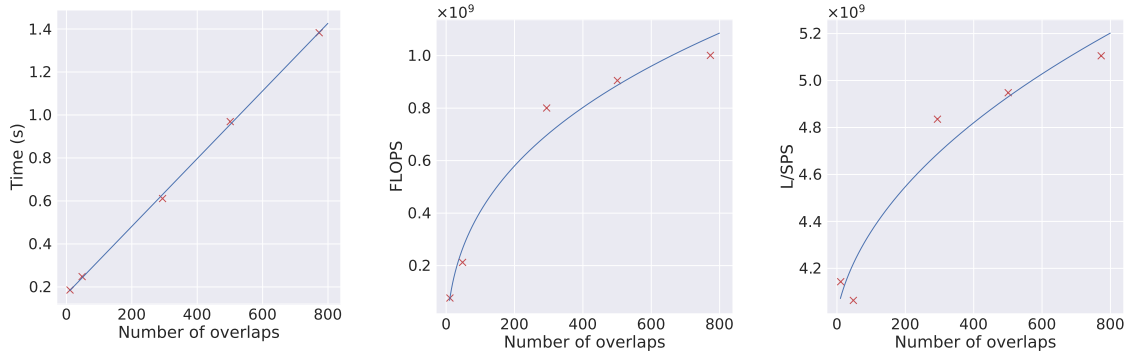
(a) Avg Time (s), R^2=0.995 (b) FLOPS, R^2=0.855 (c) L/SPS, R^2=0.514

Figure 7.1.: *Montage*: **mConcatFit** performance model by *Extra-P*

mBgModel Table 7.7 depicts the performance metrics of mBgModel jobs. This is another short job. However, it has the second-highest number of FLOPS among all jobs in the *Montage* workflow and can be considered CPU-intensive.

Instance	Time (s) Total	Avg	Min	Max	FLOPS Avg	L/Sps Avg	Memory (MB) Peak	File I/O In	Out
0	0.19	-	-	-	7.66×10^7	4.14×10^9	4	<1	<1
1	0.25	-	-	-	2.12×10^8	4.06×10^9	4	<1	<1
2	0.61	-	-	-	8.00×10^8	4.84×10^9	4	<1	<1
3	0.97	-	-	-	9.05×10^8	4.95×10^9	4	<1	<1
4	1.38	-	-	-	1.00×10^9	5.11×10^9	4	<1	<1

Table 7.7.: *Montage*: **mBgModel** job data



(a) Avg Time (s), R^2=0.999 (b) FLOPS, R^2=0.916 (c) L/SPS, R^2=0.785

Figure 7.2.: *Montage*: **mBgModel** performance model by *Extra-P*

mBackground Table 7.8 depicts the performance metrics of mBackground jobs.

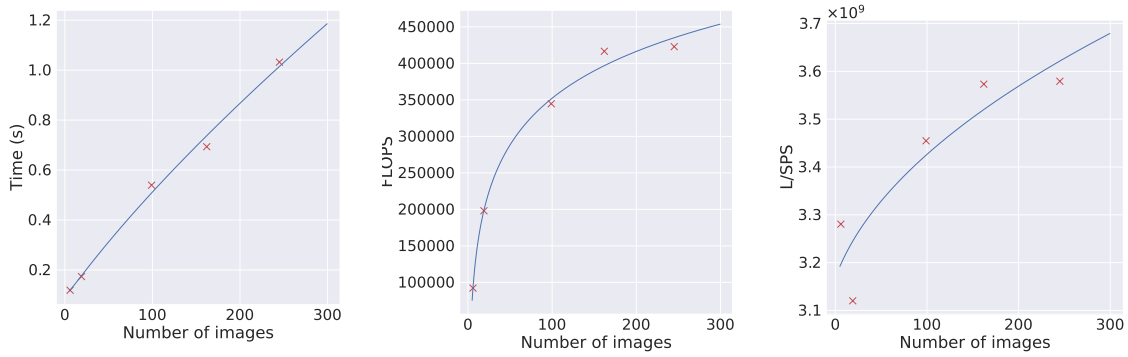
Instance	Time (s) Total	Avg	Min	Max	FLOPS Avg	L/Sps Avg	Memory (MB) Peak	File I/O In	Out
0	1.43	0.24	0.23	0.25	1.97×10^7	3.34×10^9	4	12	12
1	5.98	0.31	0.27	0.39	1.49×10^7	2.55×10^9	4	12	12
2	34.43	0.43	0.25	0.51	1.12×10^7	1.87×10^9	6	12	12
3	72.23	0.45	0.22	0.63	1.08×10^7	1.83×10^9	4	12	12
4	109.60	0.46	0.23	0.64	1.08×10^7	1.83×10^9	4	12	12

Table 7.8.: *Montage*: **mBackground** job data

mImgtbl Table 7.9 depicts the performance metrics of mImgtbl jobs.

Instance	Time (s) Total	Avg	Min	Max	FLOPS Avg	L/Sps Avg	Memory (MB) Peak	File I/O In	Out
0	0.12	-	-	-	9.21×10^4	3.28×10^9	4	64	<1
1	0.17	-	-	-	1.98×10^5	3.12×10^9	4	201	<1
2	0.54	-	-	-	3.45×10^5	3.45×10^9	4	1100	<1
3	0.69	-	-	-	4.17×10^5	3.57×10^9	4	1700	<1
4	1.03	-	-	-	4.23×10^5	3.58×10^9	4	2600	<1

Table 7.9.: *Montage*: **mImgtbl** job data



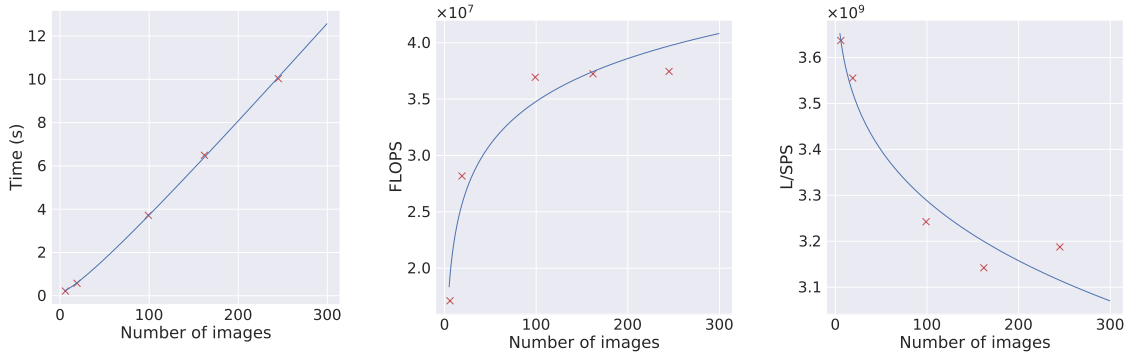
(a) Avg Time (s), R^2=0.982 (b) FLOPS, R^2=0.978 (c) L/SPS, R^2=0.238

Figure 7.3.: *Montage*: **mImgtbl** performance model by *Extra-P*

mAdd Table 7.10 depicts the performance metrics of mAdd jobs.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	0.21	-	-	-	1.71×10^7	3.64×10^9	4	64	13
1	0.58	-	-	-	2.82×10^7	3.56×10^9	4	201	52
2	3.72	-	-	-	3.69×10^7	3.24×10^9	57	1100	467
3	6.49	-	-	-	3.72×10^7	3.14×10^9	79	1700	829
4	10.03	-	-	-	3.75×10^7	3.19×10^9	81	2600	1296

Table 7.10.: *Montage*: **mAdd** job data



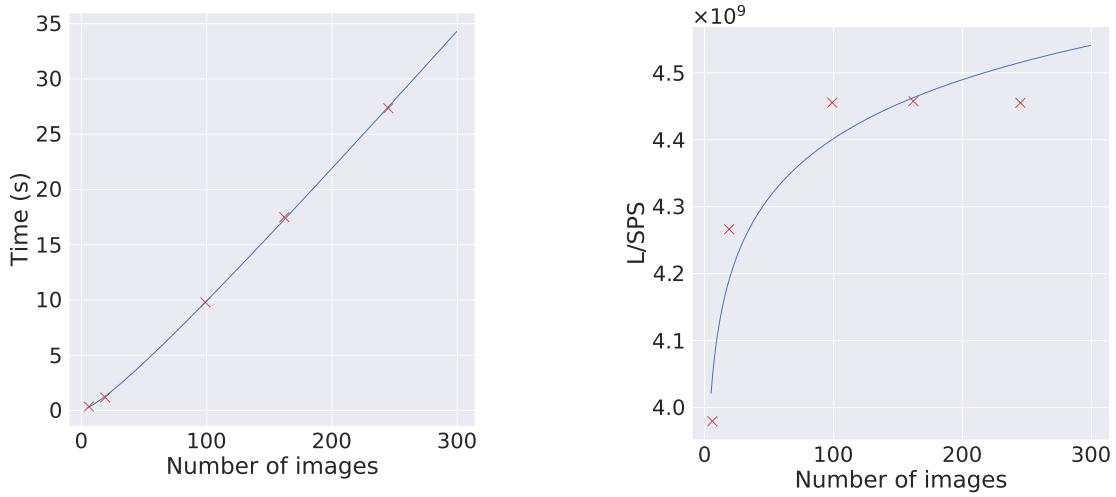
(a) Avg Time (s), R^2=0.999 (b) FLOPS, R^2=0.619 (c) L/SPS, R^2=0.780

Figure 7.4.: *Montage*: **mAdd** performance model by *Extra-P*

mViewer Table 7.11 depicts the performance metrics of mViewer jobs.

Instance	Time (s) Total	Avg	Min	Max	FLOPS Avg	L/Sps Avg	Memory (MB) Peak	File I/O In	Out
0	0.35	-	-	-	4.67×10^7	3.98×10^9	4	13	<1
1	1.18	-	-	-	2.02×10^7	4.27×10^9	4	52	1
2	9.78	-	-	-	1.04×10^7	4.46×10^9	573	467	12
3	17.49	-	-	-	9.70×10^6	4.46×10^9	950	829	21
4	27.35	-	-	-	9.40×10^6	4.46×10^9	1400	1296	34

Table 7.11.: *Montage*: **mViewer** job data



(a) Avg Time (s), R^2=0.999

(b) L/SPS, R^2=0.442

Figure 7.5.: *Montage*: **mViewer** performance model by *Extra-P*

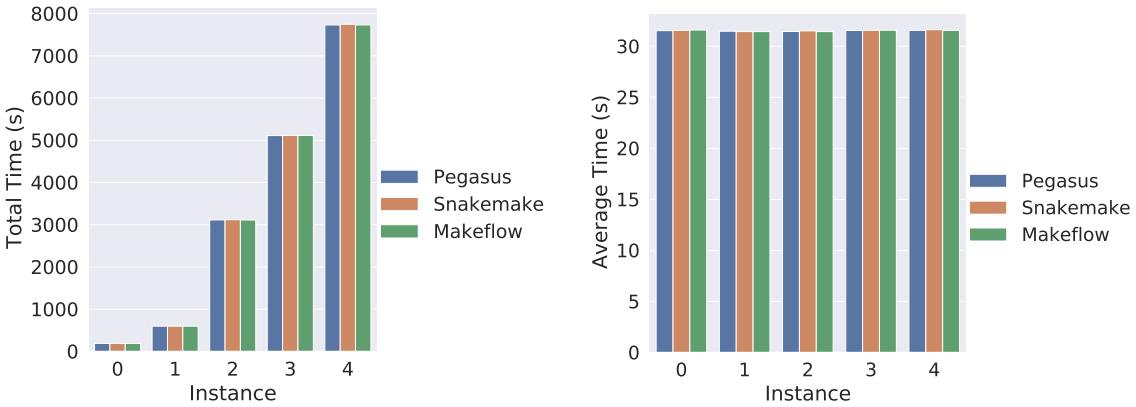
7.2.3. Workflow Management System Profiles

The following paragraphs and tables compare the execution time with all workflow management systems for the *Montage* workflow. In each table, we highlight the system with the lowest average time per instance.

mProject Table 7.12 and Figure 7.6 depict the execution times of the mProject jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	189.20	31.53	<0.01	189.37	31.56	<0.01	189.57	31.59	0.01
1	598.05	31.48	<0.01	597.44	31.44	<0.01	597.37	31.44	<0.01
2	3113.47	31.45	<0.01	3119.39	31.51	<0.01	3112.53	31.44	<0.01
3	5111.70	31.55	<0.01	5114.30	31.57	<0.01	5114.83	31.57	<0.01
4	7731.90	31.56	<0.01	7747.75	31.62	<0.01	7731.94	31.56	<0.01

Table 7.12.: *Montage*: **mProject** execution time with WMSs



(a) Total Time (s)

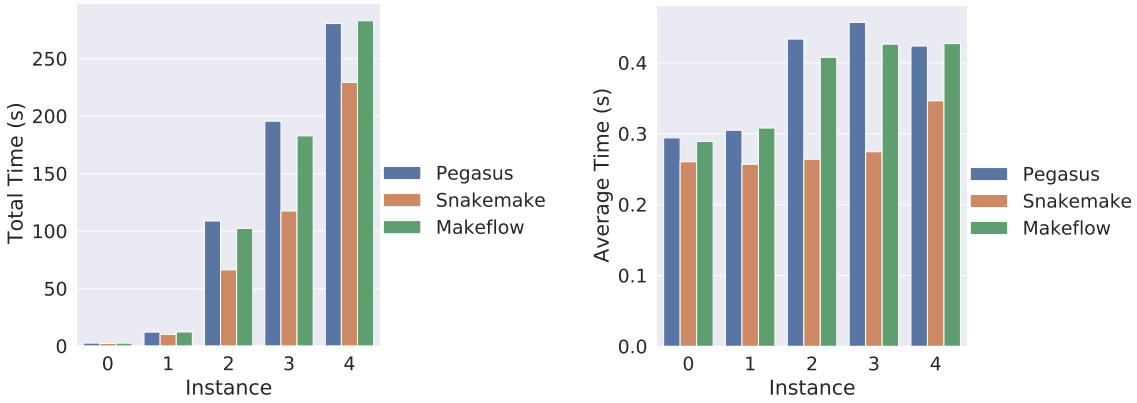
(b) Average Time (s)

Figure 7.6.: *Montage*: **mProject** WMS comparison

mDiffFit Table 7.13 and Figure 7.7 depict the execution times of the mDiffFit jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	2.65	0.29	0.10	2.34	0.26	0.09	2.60	0.29	0.09
1	12.20	0.30	0.03	10.28	0.26	0.07	12.32	0.31	0.03
2	108.83	0.43	0.09	66.28	0.26	0.05	102.37	0.41	0.03
3	195.61	0.46	0.04	117.52	0.27	0.05	182.84	0.43	0.05
4	280.52	0.42	0.07	229.31	0.35	0.14	282.96	0.43	0.07

Table 7.13.: *Montage*: **mDiffFit** execution time with WMSs



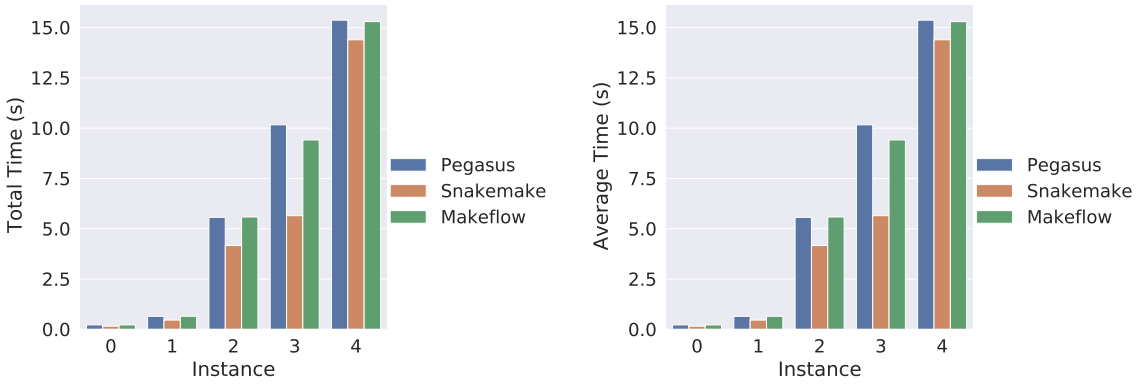
(a) Total Time (s) (b) Average Time (s)

Figure 7.7.: *Montage*: **mDiffFit** WMS comparison

mConcatFit Table 7.14 and Figure 7.8 depict the execution times of the mConcatFit jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	0.22	0.22	0.13	0.16	0.16	0.08	0.22	0.22	0.23
1	0.65	0.65	0.16	0.46	0.46	0.21	0.65	0.65	0.10
2	5.57	5.57	0.03	4.17	4.17	0.26	5.59	5.59	0.09
3	10.17	10.17	0.03	5.65	5.65	0.22	9.41	9.41	0.05
4	15.37	15.37	0.06	14.39	14.39	0.06	15.29	15.29	0.11

Table 7.14.: *Montage*: **mConcatFit** execution time with WMSs



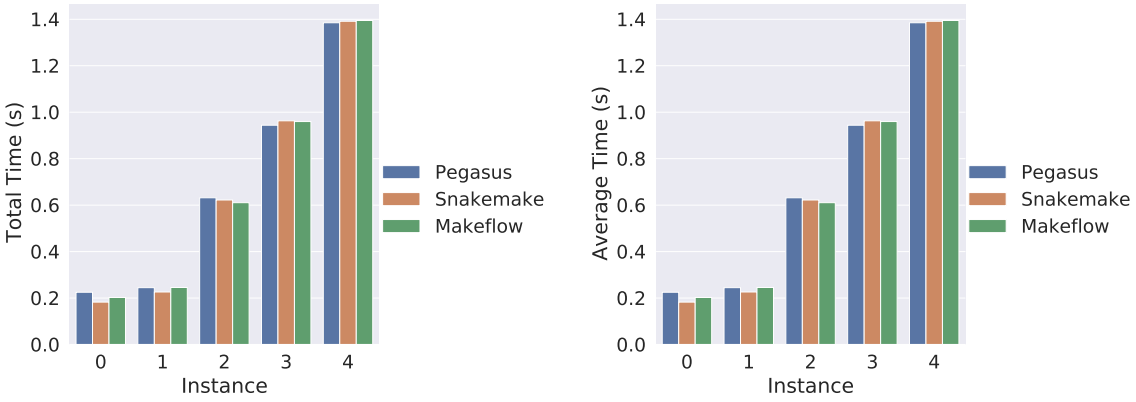
(a) Total Time (s) (b) Average Time (s)

Figure 7.8.: *Montage*: **mConcatFit** WMS comparison

mBgModel Table 7.15 and Figure 7.9 depict the execution times of the mBgModel jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	0.23	0.23	0.21	0.18	0.18	0.05	0.20	0.20	0.15
1	0.24	0.24	0.06	0.23	0.23	0.06	0.25	0.25	0.10
2	0.63	0.63	0.02	0.62	0.62	0.06	0.61	0.61	0.02
3	0.94	0.94	0.02	0.96	0.96	0.03	0.96	0.96	0.01
4	1.38	1.38	0.01	1.39	1.39	0.01	1.39	1.39	0.01

Table 7.15.: *Montage*: **mBgModel** execution time with WMSs



(a) Total Time (s)

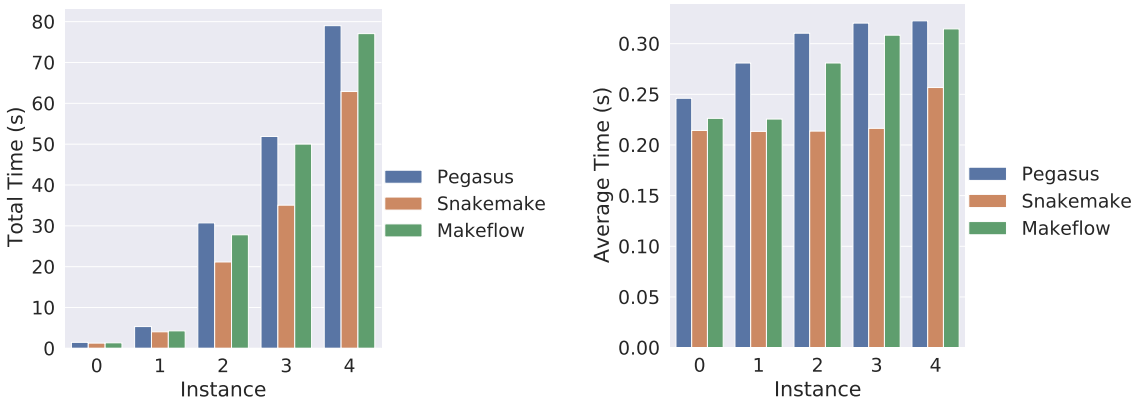
(b) Average Time (s)

Figure 7.9.: *Montage*: **mBgModel** WMS comparison

mBackground Table 7.16 and Figure 7.10 depict the execution times of the mBackground-jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	1.48	0.25	0.12	1.29	0.21	0.01	1.36	0.23	0.02
1	5.34	0.28	0.08	4.05	0.21	0.01	4.29	0.23	0.02
2	30.72	0.31	0.01	21.16	0.21	<0.01	27.81	0.28	0.08
3	51.89	0.32	0.02	35.06	0.22	0.01	50.01	0.31	0.06
4	79.02	0.32	0.05	62.89	0.26	0.01	77.07	0.31	0.02

Table 7.16.: *Montage*: **mBackground** execution time with WMSs



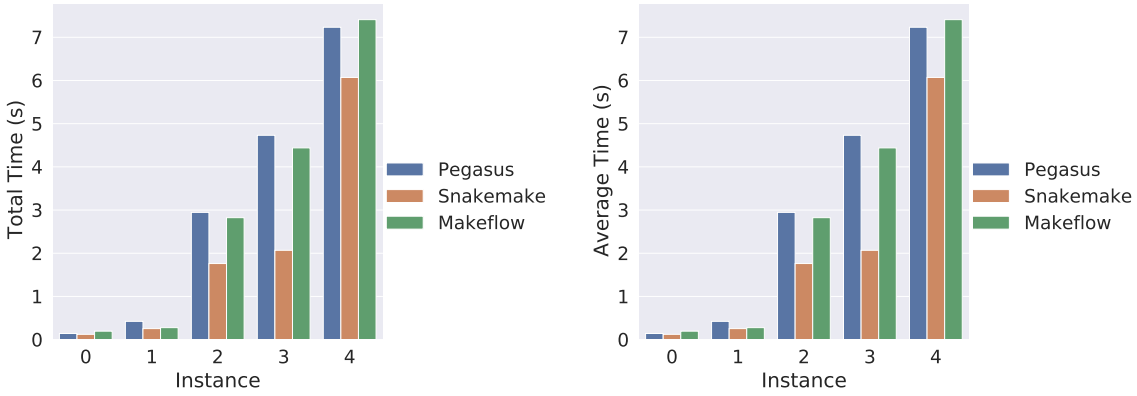
(a) Total Time (s) (b) Average Time (s)

Figure 7.10.: *Montage*: **mBackground** WMS comparison

mImgtbl Table 7.17 and Figure 7.11 depict the execution times of the mImgtbl jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	0.14	0.14	0.08	0.12	0.12	0.11	0.20	0.20	0.66
1	0.42	0.42	0.38	0.26	0.26	0.48	0.28	0.28	0.15
2	2.95	2.95	0.13	1.76	1.76	0.63	2.82	2.82	0.34
3	4.73	4.73	0.09	2.07	2.07	0.60	4.44	4.44	0.07
4	7.23	7.23	0.09	6.07	6.07	0.03	7.41	7.41	0.03

Table 7.17.: *Montage*: **mImgtbl** execution time with WMSs



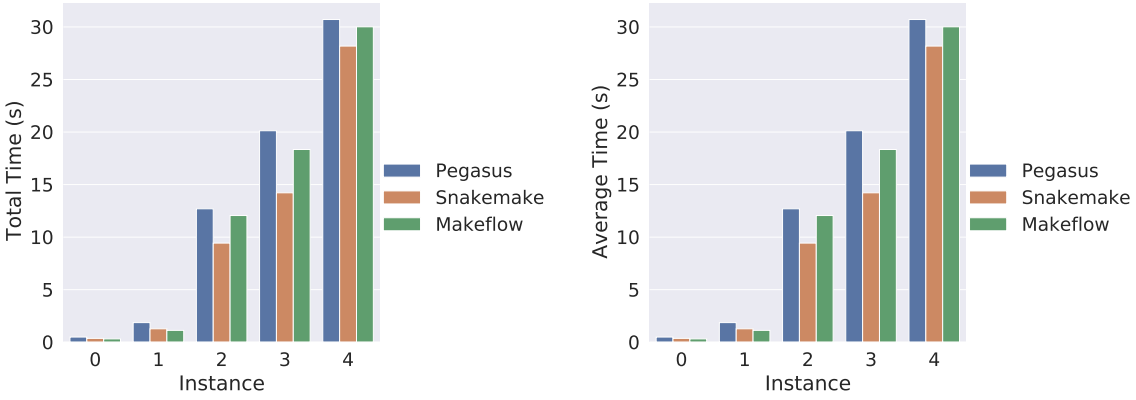
(a) Total Time (s) (b) Average Time (s)

Figure 7.11.: *Montage*: **mImgtbl** WMS comparison

mAdd Table 7.18 and Figure 7.12 depict the execution times of the mAdd jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	0.49	0.49	0.20	0.37	0.37	0.24	0.32	0.32	0.15
1	1.87	1.87	0.21	1.28	1.28	0.30	1.13	1.13	0.30
2	12.70	12.70	0.12	9.43	9.43	0.25	12.06	12.06	0.16
3	20.12	20.12	0.08	14.22	14.22	0.25	18.34	18.34	0.05
4	30.71	30.71	0.03	28.18	28.18	0.02	30.01	30.01	0.06

Table 7.18.: *Montage*: **mAdd** execution time with WMSs



(a) Total Time (s)

(b) Average Time (s)

Figure 7.12.: *Montage*: **mAdd** WMS comparison

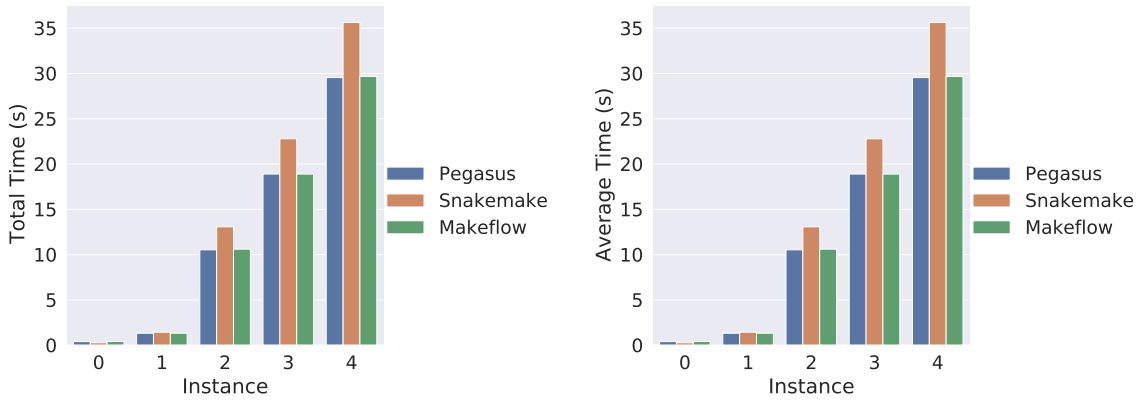
mViewer Table 7.19 and Figure 7.13 depict the execution times of the mViewer jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	0.40	0.40	0.01	0.27	0.27	0.03	0.41	0.41	0.04
1	1.33	1.33	0.01	1.44	1.44	0.03	1.33	1.33	0.04
2	10.53	10.53	<0.01	13.07	13.07	0.02	10.61	10.61	<0.01
3	18.90	18.90	<0.01	22.79	22.79	<0.01	18.90	18.90	0.01
4	29.55	29.55	<0.01	35.63	35.63	<0.01	29.67	29.67	0.01

Table 7.19.: *Montage*: **mViewer** execution time with WMSs

Instance ID	individuals	individuals_merge	sifting	mutation_overlap	frequency	total jobs
0	2	1	1	7	7	18
1	4	1	1	7	7	20
2	10	1	1	7	7	26
3	20	1	1	7	7	36
4	40	1	1	7	7	56

Table 7.20.: Numbers of jobs by job type for all *1000Genome* instances



(a) Total Time (s)

(b) Average Time (s)

Figure 7.13.: *Montage*: **mViewer** WMS comparison

7.3. 1000Genome

This section presents the performance evaluation of the *1000Genome* workflow. We execute all scripts with *Python* 3.6.8.

7.3.1. Input Instances

We profile the *1000Genome* workflow with five different input instances. For the different instances, we do not vary the size of the input but increase the number of parallel jobs that parse the chromosome data. Table 7.20 shows the numbers of jobs by job type for all instances.

7.3.2. Characterization

This section characterizes the *1000Genome* workflow regarding resource utilization. All *1000Genome* instances are executed on cluster nodes with 180 GB of memory.

7.3.2.1. Workflow Level

Table 7.21 depicts general data of the workflow runs for all five input instances. Note that the walltime numbers contain an overhead of roughly 30 seconds due to the Slurm batch system. The runtime of *1000Genome* is primarily influenced by the degree of parallelism in the individuals jobs. However, the more individuals jobs we use, the higher the peak memory usage is.

Instance ID	Total Jobs	Workflow Walltime (s)	Peak Memory Usage (MB)
0	18	36 353.2	11 656.0
1	20	18 616.4	16 402.0
2	26	8299.0	30 732.0
3	36	4801.9	54 589.0
4	56	3472.0	102 343.0

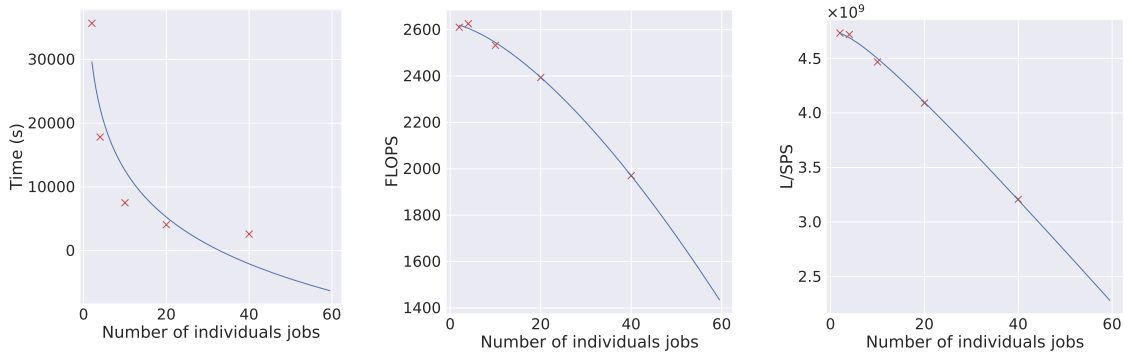
Table 7.21.: *1000Genome* Workflow Level

The following paragraphs and tables examine the *1000Genome* workflow on a job level by observing the isolated performance metrics of all jobs independently.

individuals Table 7.22 and Figure 7.14 depict performance metrics of individuals jobs.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	71 343.45	35 671.72	35 521.49	35 821.96	2610.51	4.73×10^9	6017	2539	93
1	71 290.47	17 822.61	17 595.38	18 070.41	2625.63	4.72×10^9	4238	2539	47
2	75 067.81	7506.78	7404.28	7567.83	2532.38	4.47×10^9	3170	2539	16
3	81 470.26	4073.51	3977.91	4152.27	2392.50	4.09×10^9	2814	2539	6
4	103 913.25	2597.83	2480.10	2669.64	1970.65	3.21×10^9	2641	2539	2

Table 7.22.: **individuals** job data



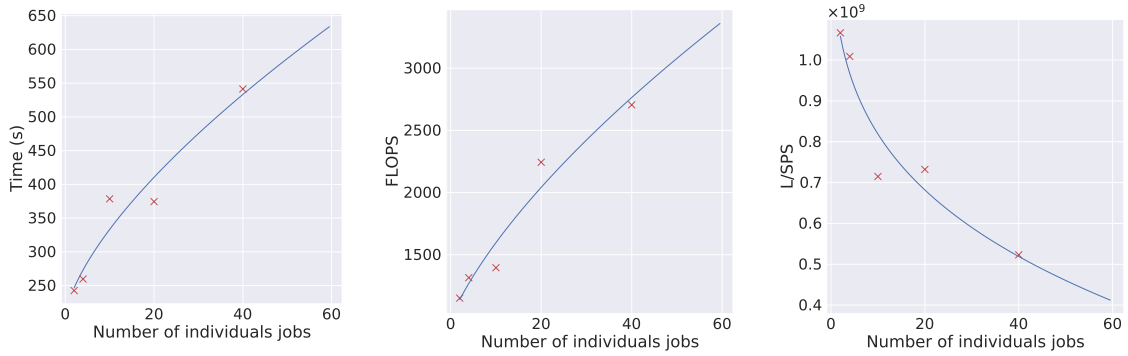
(a) Avg Time (s), R^2=0.230 (b) FLOPS, R^2=0.993 (c) L/SPS, R^2=0.994

Figure 7.14.: *1000Genome*: **individuals** performance model by *Extra-P*

individuals_merge Table 7.23 and Figure 7.15 depict performance metrics of individuals_merge jobs.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	242.64	-	-	-	1150.79	1.07×10^9	1670	185	184
1	259.75	-	-	-	1315.27	1.01×10^9	1670	186	184
2	378.57	-	-	-	1395.18	7.15×10^8	1672	160	184
3	374.47	-	-	-	2243.22	7.32×10^8	1680	117	184
4	541.34	-	-	-	2704.84	5.23×10^8	1674	82	184

Table 7.23.: **individuals_merge** job data



(a) Avg Time (s), R^2=0.831 (b) FLOPS, R^2=0.845 (c) L/SPS, R^2=0.828

Figure 7.15.: *1000Genome*: **individuals_merge** performance model by *Extra-P*

other jobs Since all sifting, mutation_overlap and frequency jobs in our choice of instances use the same inputs and sizes, we choose to only show the values for the instances with the lowest CoV. The peak memory values for sifting jobs have too high variance, so we omitted them.

		Time (s)					FLOPS	
Job	Instance	Total	Avg	Min	Max	CoV	Avg	CoV
sifting	3	4.00	-	-	-	0.08	67 875	0.08
mutation_overlap	4	130.37	18.62	11.71	43.67	0.01	3.61×10^6	0.01
frequency	1	1069.58	152.80	131.41	181.96	<0.01	1.52×10^6	<0.01

Table 7.24.: **other** job data

Job	Instance	L/SPS Avg	L/SPS CoV	Memory (MB) Peak	File I/O In	File I/O Out
sifting	2	8.81×10^8	0.08	-	1580	2
mutation_overlap	4	1.03×10^9	0.01	152	184	17
frequency	1	1.80×10^9	<0.01	170	184	1

Table 7.25.: **other** job data

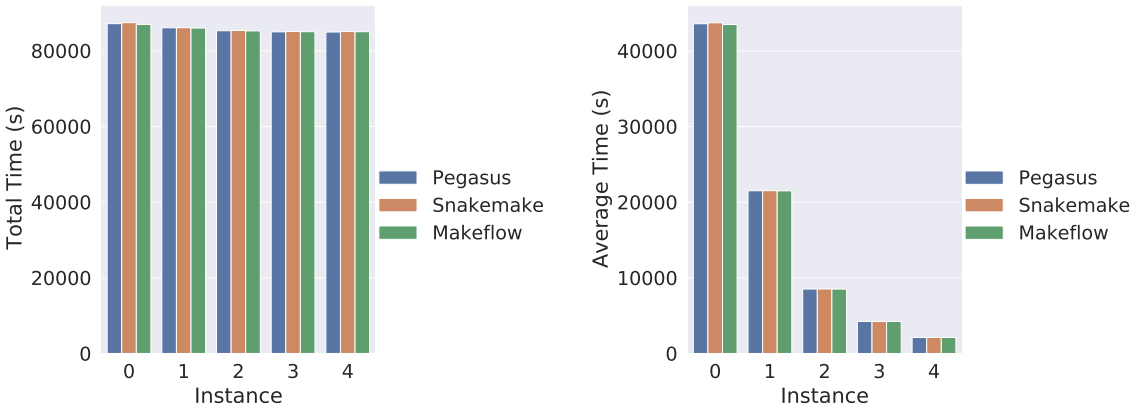
7.3.3. Workflow Management System Profiles

The following paragraphs and tables compare the execution time with all workflow management systems for the *1000Genome* workflow. In each table, we highlight the system with the lowest average time per instance.

individuals Table 7.26 and Figure 7.16 depict the execution times of the individuals jobs.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	87 194.70	43 597.35	<0.01	87 449.89	43 724.95	<0.01	86 991.98	43 495.99	<0.01
1	86 095.58	21 523.89	<0.01	86 128.50	21 532.13	<0.01	86 035.68	21 508.92	<0.01
2	85 319.39	8531.94	<0.01	85 392.58	8539.26	<0.01	85 273.12	8527.31	<0.01
3	85 024.73	4251.24	<0.01	85 115.27	4255.76	<0.01	85 065.07	4253.25	<0.01
4	84 978.91	2124.47	<0.01	85 111.25	2127.78	<0.01	85 060.58	2126.51	<0.01

Table 7.26.: *1000Genome*: **individuals** execution time with WMSs



(a) Total Time (s)

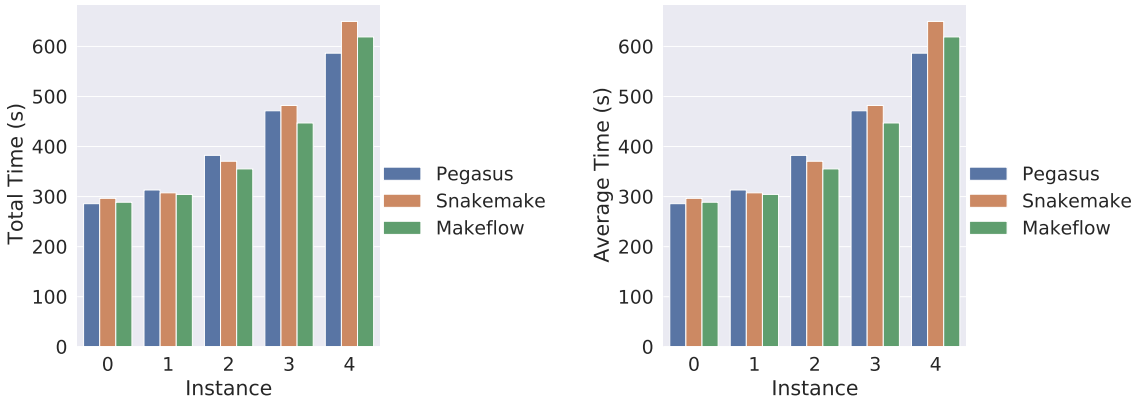
(b) Average Time (s)

Figure 7.16.: *1000Genome*: **individuals** WMS comparison

individuals_merge Table 7.27 and Figure 7.17 depict the execution times of the individuals_merge jobs. For higher instances, the average time of executions with *Snakemake* increases more than those with other systems.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	285.83	285.83	0.01	296.55	296.55	0.02	288.73	288.73	0.02
1	313.18	313.18	0.01	307.51	307.51	0.03	304.37	304.37	0.01
2	382.41	382.41	0.05	370.61	370.61	0.03	355.30	355.30	0.01
3	471.44	471.44	0.07	482.05	482.05	0.07	447.10	447.10	0.02
4	586.54	586.54	0.03	649.93	649.93	0.05	618.99	618.99	0.07

Table 7.27.: *1000Genome*: **individuals_merge** execution time with WMSs



(a) Total Time (s)

(b) Average Time (s)

Figure 7.17.: *1000Genome*: **individuals_merge** WMS comparison

7.4. Orcasound

This section presents the performance evaluation of the *Orcasound* workflow. We execute all scripts with *Python* 3.6.8.

7.4.1. Input Instances

We profile the *Orcasound* workflow with five different input instances of varying duration. Table 7.28 shows the individual characteristics of the input instances that we use for the *Orcasound* workflow. For each instance, we double the amount of timestamps we use as input. The audio batches of each timestamp do not have the exact same runtime, but each instance roughly doubles the length of the previous one. Table 7.29 presents a more

detailed breakdown of the number of jobs of each type per instance. Since instance 0 only has one inference job, we do not need a merge job in this case.

Instance ID	Number of Timestamps	Begin TS	End TS	Total length
0	1	1601188222	1601188222	6 hours
1	2	1601188222	1601209820	12 hours
2	4	1601188222	1601253021	24 hours
3	8	1601188222	1601339419	48 hours
4	16	1601188222	1601512219	96 hours

Table 7.28.: Input instances of the *Orcasound* workflow

	Instance ID				
Job	0	1	2	3	4
convert2wav	1	2	4	8	16
convert2spectrogram	1	2	4	8	16
inference	1	2	4	8	16
merge	0	1	1	1	1
total jobs	3	7	13	25	49

Table 7.29.: Numbers of jobs by job type for all *Orcasound* instances

7.4.2. Characterization

This section characterizes the *Orcasound* workflow regarding resource utilization.

Workflow Level Table 7.3 depicts general data of the workflow runs for all five input instances. Note that the walltime numbers contain an overhead of roughly 30 seconds due to the Slurm batch system. The memory and walltime numbers are all averaged over five different runs.

Instance ID	Total Jobs	Workflow Walltime (s)	Peak Memory Usage (MB)
0	3	1143.5	604.4
1	7	1169.0	1075.0
2	13	1208.7	2043.0
3	25	1309.9	3971.0
4	49	1561.2	7800.0

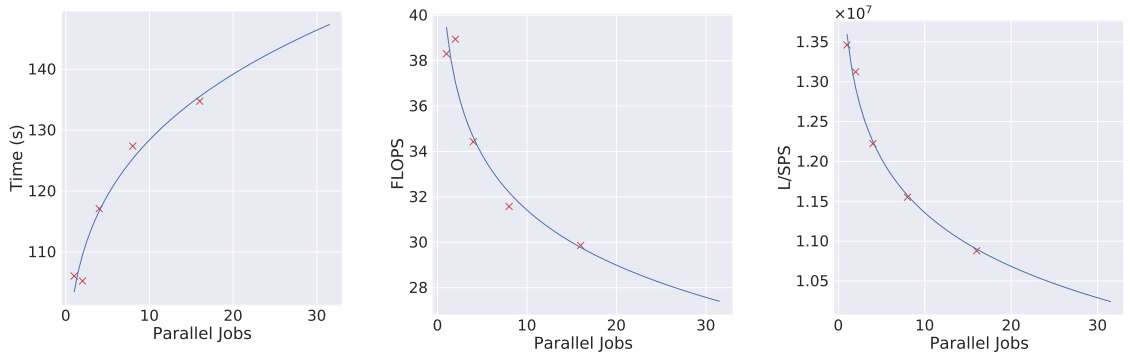
Table 7.30.: *Orcasound* Workflow Level

The following paragraphs and tables examine the *Orcasound* workflow on a job level by observing the isolated performance metrics of all jobs independently.

convert2wav Table 7.31 and Figure 7.18 depict the performance metrics of convert2wav jobs. These jobs have a low number of under 40 FLOPS. They read thousands of small sound files each and output them in a converted format. Consequently, this job type can be considered I/O-intensive. Increasing the number of parallel jobs reduces the performance.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	106.04	106.04	106.04	106.04	38.30	1.35×10^7	98	223	2000
1	210.45	105.22	104.27	106.19	38.94	1.31×10^7	96	223	2000
2	468.45	117.11	111.88	120.14	34.43	1.22×10^7	98	223	2000
3	1019.07	127.38	123.95	130.28	31.58	1.16×10^7	98	222	2000
4	2156.02	134.75	127.80	138.72	29.86	1.09×10^7	98	221	2000

Table 7.31.: *Orcasound*: **convert2wav** job data



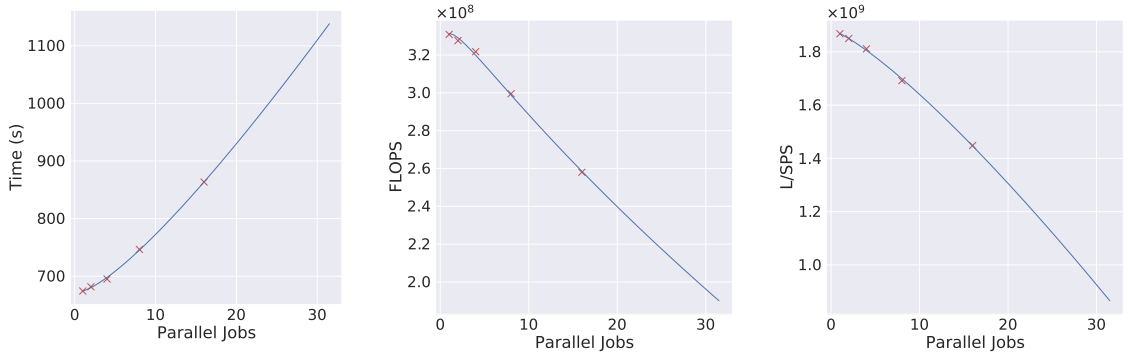
(a) Avg Time (s), R^2=0.835 (b) FLOPS, R^2=0.652 (c) L/SPS, R^2=0.942

Figure 7.18.: *Orcasound*: **convert2wav** performance model by *Extra-P*

convert2spectrogram Table 7.32 and Figure 7.19 depict the performance metrics of convert2spectrogram jobs. This job takes up the largest portion of the *Orcasound* runtime. It is both CPU- and I/O-intensive. Increasing the number of parallel jobs reduces the performance.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	674.34	674.34	674.34	674.34	3.31×10^8	1.87×10^9	222	2000	311
1	1363.59	681.79	680.40	683.19	3.28×10^8	1.85×10^9	224	2000	310
2	2780.95	695.24	690.05	702.67	3.22×10^8	1.81×10^9	223	2000	308
3	5972.76	746.60	738.49	756.88	3.00×10^8	1.69×10^9	221	2000	307
4	13 815.74	863.48	854.19	872.94	2.58×10^8	1.45×10^9	222	2000	307

Table 7.32.: *Orcasound*: **convert2spectrogram** job data



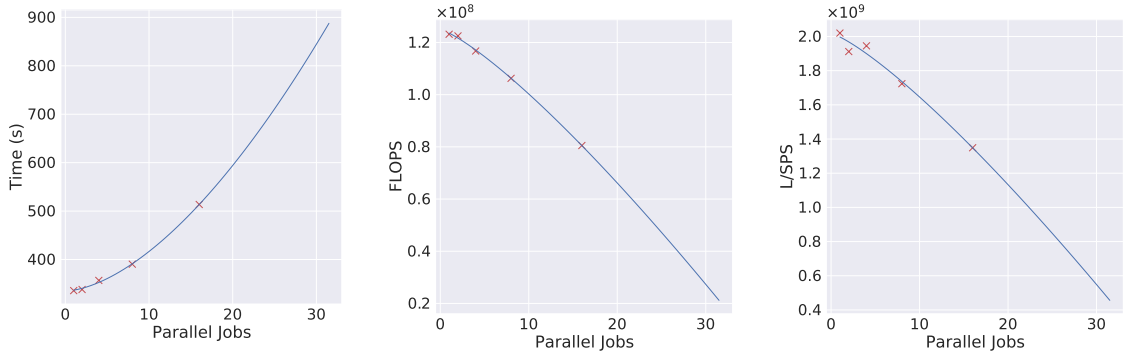
(a) Avg Time (s), R^2=0.999 (b) FLOPS, R^2=0.990 (c) L/SPS, R^2=0.995

Figure 7.19.: *Orcasound*: **convert2spectrogram** performance model by *Extra-P*

inference Table 7.33 and Figure 7.20 depict the performance metrics of inference jobs. Inference jobs have the highest peak memory usage of all *Orcasound* jobs. They read around 2 GB of data each, but output only a small file. Both the number of FLOPS and L/SPS are above average. As with both other parallel jobs, the performance decreases with more jobs running in parallel.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
0	335.73	335.73	335.73	335.73	1.23×10^8	2.02×10^9	661	2000	<1
1	676.17	338.08	336.78	339.39	1.23×10^8	1.91×10^9	663	2000	<1
2	1427.35	356.84	353.76	360.71	1.17×10^8	1.95×10^9	663	2000	<1
3	3123.20	390.40	384.65	398.34	1.06×10^8	1.72×10^9	663	2000	<1
4	8217.46	513.59	487.89	524.58	8.05×10^7	1.35×10^9	662	2000	<1

Table 7.33.: *Orcasound*: **inference** job data



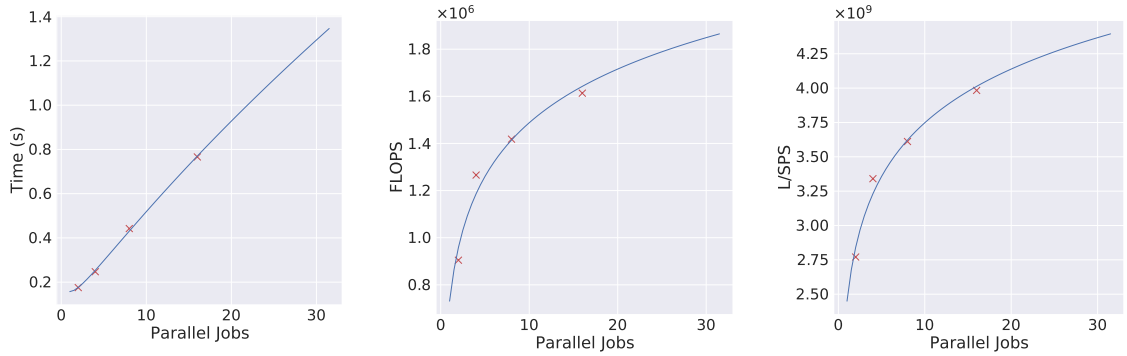
(a) Avg Time (s), R^2=0.997 (b) FLOPS, R^2=0.998 (c) L/SPS, R^2=0.945

Figure 7.20.: *Orcasound*: **inference** performance model by *Extra-P*

merge Table 7.34 and Figure 7.21 depict the performance metrics of merge jobs. Since this is a data aggregation job, it is the only *Orcasound* task which is not executed in parallel. It also has the highest number of L/SPS and can be considered I/O intensive.

	Time (s)				FLOPS	L/Sps	Memory (MB)	File I/O	
Instance	Total	Avg	Min	Max	Avg	Avg	Peak	In	Out
1	0.18	-	-	-	9.05×10^5	2.77×10^9	4	<1	<1
2	0.25	-	-	-	1.27×10^6	3.34×10^9	4	1	1
3	0.44	-	-	-	1.42×10^6	3.61×10^9	4	3	3
4	0.77	-	-	-	1.61×10^6	3.98×10^9	4	5	5

Table 7.34.: *Orcasound*: **merge** job data



(a) Avg Time (s), R^2=0.990 (b) FLOPS, R^2=0.696 (c) L/SPS, R^2=0.831

Figure 7.21.: *Orcasound*: **merge** performance model by *Extra-P*

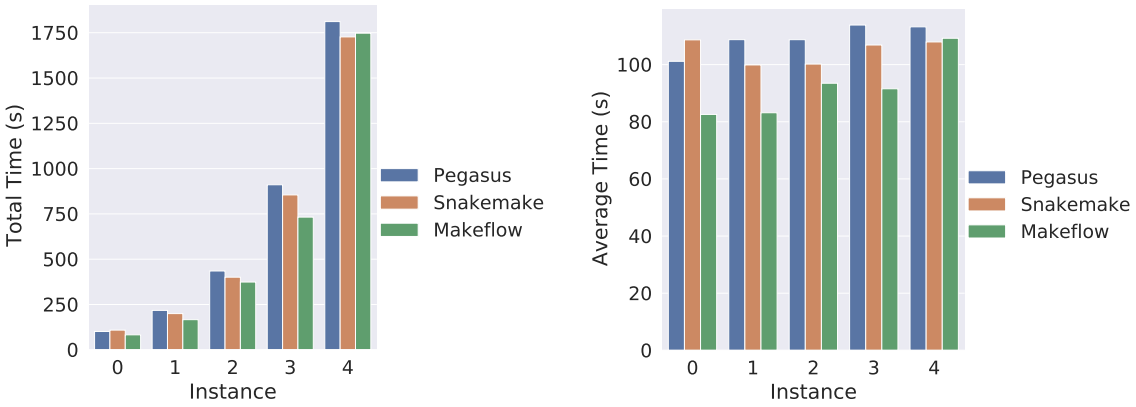
7.4.3. Workflow Management System Profiles

The following paragraphs and tables compare the execution time with all workflow management systems for the *Orcasound* workflow. In each table, we highlight the system with the lowest average time per instance.

convert2wav Table 7.35 and Figure 7.22 depict the execution times of the convert2wav jobs. Apart from instance 0, *Pegasus* has the highest average time for this job type. Executions with *Makeflow* have the lowest average time for all instances except the last one.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	101.16	101.16	0.13	108.66	108.66	0.17	82.57	82.57	0.07
1	217.47	108.74	0.01	199.77	99.89	0.01	166.31	83.15	0.05
2	434.90	108.73	0.02	400.77	100.19	0.08	373.93	93.48	0.07
3	910.80	113.85	0.06	854.88	106.86	0.05	732.35	91.54	0.05
4	1811.50	113.22	0.04	1726.93	107.93	0.05	1747.56	109.22	0.04

Table 7.35.: *Orcasound:* **convert2wav** execution time with WMSs



(a) Total Time (s)

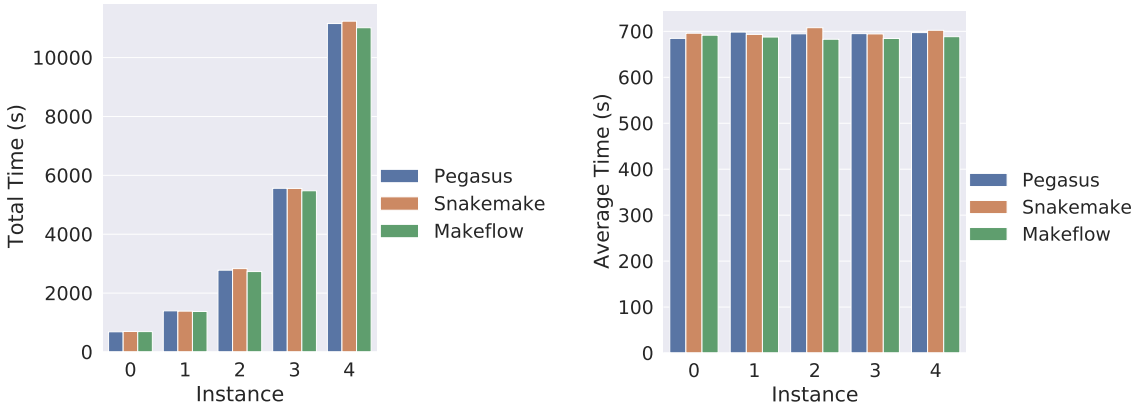
(b) Average Time (s)

Figure 7.22.: *Orcasound:* **convert2wav** WMS comparison

convert2spectrogram Table 7.36 and Figure 7.23 depict the execution times of the convert2spectrogram jobs. For this job type, *Makeflow* offers the lowest execution time for all instances after instance 0. However, the differences in time are not significant.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	684.71	684.71	0.01	695.89	695.89	0.04	691.63	691.63	0.01
1	1396.80	698.40	0.03	1386.65	693.32	0.01	1375.191	687.59	0.01
2	2778.47	694.62	0.01	2831.78	707.94	0.03	2731.32	682.83	0.01
3	5560.20	695.03	0.01	5555.06	694.38	<0.01	5475.73	684.47	0.01
4	11 156.45	697.28	0.01	11 237.08	702.32	<0.01	11 014.18	688.39	0.01

Table 7.36.: *Orcasound:* **convert2spectrogram** execution time with WMSs



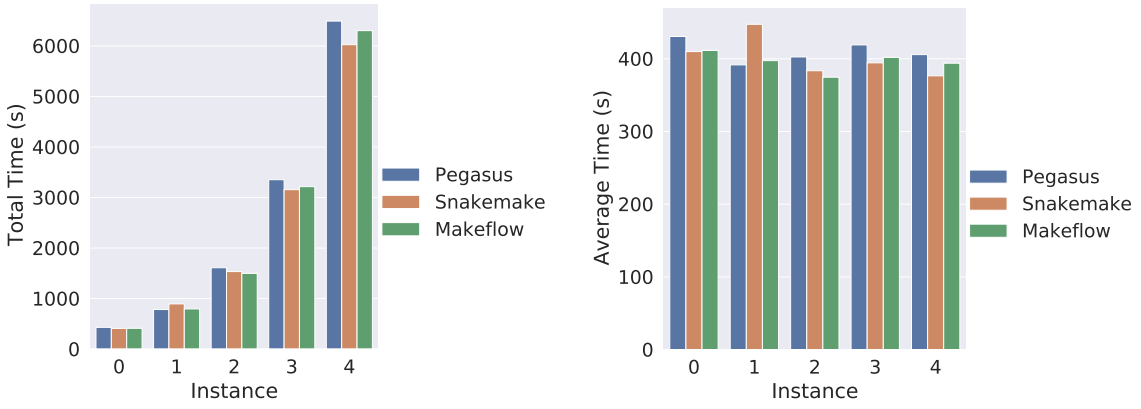
(a) Total Time (s) (b) Average Time (s)

Figure 7.23.: *Orcasound:* **convert2spectrogram** WMS comparison

inference Table 7.37 and Figure 7.24 depict the execution times of the inference jobs. Generally, *Snakemake* offers the lowest execution time, but there is no clear performance advantage over all instances.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
0	430.64	430.64	0.11	409.89	409.89	0.07	411.45	411.45	0.05
1	783.21	391.60	0.06	894.75	447.38	0.03	794.91	397.46	0.03
2	1610.04	402.51	0.05	1534.18	383.55	0.06	1498.47	374.62	0.01
3	3352.33	419.04	0.06	3155.15	394.39	0.02	3215.1	401.89	0.06
4	6491.08	405.69	0.03	6024.46	376.53	0.01	6302.33	393.90	0.04

Table 7.37.: *Orcasound:* **inference** execution time with WMSs



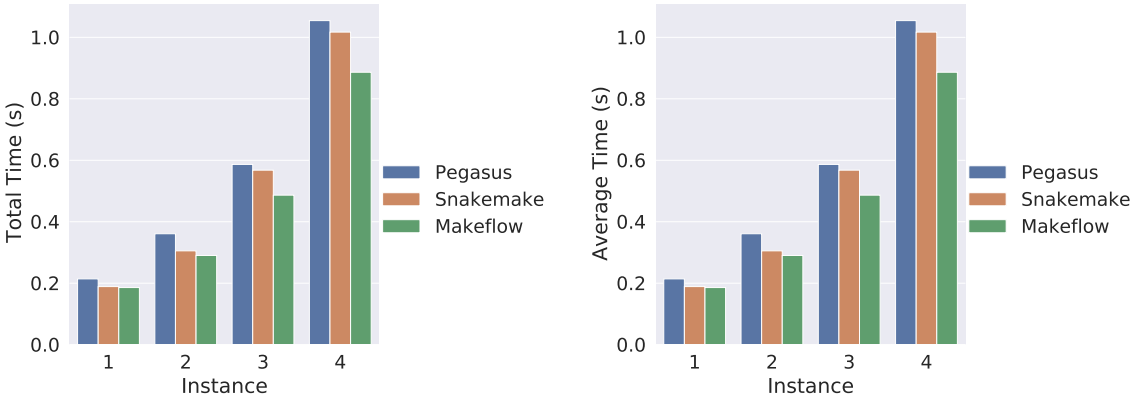
(a) Total Time (s) (b) Average Time (s)

Figure 7.24.: *Orcasound:* **inference** WMS comparison

merge Table 7.38 and Figure 7.25 depict the execution times of the merge jobs. For this job type, the systems perform equally, with a slight time advantage on the side of *Makeflow*.

Instance	Time (s)								
	Pegasus			**Snakemake**			**Makeflow**		
	Total	Avg	CoV	Total	Avg	CoV	Total	Avg	CoV
1	0.21	0.21	0.06	0.19	0.19	0.09	0.19	0.19	0.11
2	0.36	0.36	0.13	0.31	0.31	0.08	0.29	0.29	0.22
3	0.59	0.59	0.08	0.57	0.57	0.09	0.49	0.49	0.20
4	1.05	1.05	0.06	1.01	1.01	0.07	0.89	0.89	0.07

Table 7.38.: *Orcasound:* **merge** execution time with WMSs



(a) Total Time (s)

(b) Average Time (s)

Figure 7.25.: *Orcasound:* **merge** WMS comparison

7.5. Threats to Validity

While evaluating the validity of the results of this thesis, a number of internal and external threats have to be considered. Our performance profiling relies on the correctness of the metrics measured by *Score-P* and *PAPI*. Another internal threat to validity is the parallel file system which is present on the cluster. Since we store input, intermediate and output data on this shared file system, high traffic caused by other users can negatively impact the I/O throughput of the workflow tasks. Furthermore, the workflow systems we choose to compare use fundamentally different execution engines. *Pegasus*, for example, always uses HTCondor DAGman [18] to interface with a cluster, even when the cluster uses a different batch system like Slurm [10]. In contrast, *Snakemake* and *Makeflow* use special built-in executors to interface with Slurm or other batch systems [20, 2]. These different approaches could affect the measured metrics.

For external threats to validity, we have to consider that the workflows we chose may not be representative of the variety of workflow classes that are used in real science applications. This makes a generalization of our results difficult. Another impediment to the generalization of our findings are the different workflow composition concepts of the management systems: We have to create a custom definition for each workflow and system combination. This may lead to hidden optimizations that we have unintentionally included in the definition.

8. Discussion

In this chapter, we discuss the features, execution procedures and performance profiles of the different workflow management systems from the perspective of a scientist looking for an appropriate system for their research. Section 8.1 presents the goal question metric plan we use to answer our research questions. This plan lists the primary goals of the thesis and the questions we answer in order to achieve these goals.

8.1. Goal Question Metric Plan

G1: Analyze scientific workflows to characterize them with regard to their resource requirements and resource utilization from a performance perspective.
(Research Question 1)

- **Q1.1:** How do individual workflow tasks utilize the CPU?
  - **M1.1.1:** Total and average execution time
  - **M1.1.2:** Floating point operations per second (FLOPS)
- **Q1.2:** How much main memory does a workflow task require?
  - **M1.2.1:** Peak main memory usage
- **Q1.3:** How many I/O operations does a workflow task perform?
  - **M1.3.1:** Load/Store instructions per second (L/SPS)
  - **M1.3.2:** Size of input and output files

G2: Compare workflow management systems with regard to performance to help scientists choose an appropriate system for their individual needs.
(Research Questions 2 and 3)

- **Q2.1:** How fast does a workflow complete when executed with different workflow management systems?
  - **M2.2.1:** Total and average execution time
- **Q2.2:** How do workflows tasks utilize the CPU when executed with different workflow management systems?
  - **M2.2.1:** Floating point operations per second (FLOPS)

8.2. Comparison of WMS Features

This section discusses the features of workflow management systems, as detailed in chapter 5. These characteristics of WMSs are not directly related to their performance but can still play an important role in the decision-making process.

Level of Abstraction The level of abstraction provided by an abstract workflow definition is very important when defining workflows that need to be executed in different execution environments. It also allows for more changes to the workflow itself, if, for example, the workflow is still under development while writing the abstract definition. In this regard, *Pegasus* offers the highest level of abstraction because of its stringent separation of abstract and concrete workflows [9]. Through the use of catalogs, a *Pegasus* user can change everything, from environment variables to the execution engine, without editing the abstract workflow definition. In comparison, *Makeflow* offers a very low level of abstraction concerning aspects like logical filenames. If the location of data in the working directory of a workflow changes, these changes have to be made to the *Makflow* definition too.

Structural Flexibility The structures of workflows can have varying complexity. In practice, the ability to define complex behaviour in a concise way is important when writing workflow definitions for a workflow management system. When comparing the flexibility of the three WMSs we examine in this thesis, we find that *Snakemake* offers the highest structural flexibility. The concept of wildcards allows for short and generalized rules which then can represent thousands of actual tasks. *Makeflow*, on the other hand, offers the lowest structural flexibility: Since every task in *Makeflow* requires a separate rule, writing the workflow by hand becomes impossible for large workflows. In this case the user is required to write a custom script to generate the workflow definition automatically. For *Pegasus*, the advantages of catalogs and abstraction become a disadvantage in terms of simplicity: even for small workflows, the pegasus definition code is substantially larger, which can be seen in chapter 5.

Workflow Execution The ability to execute workflows on a variety of execution environments is especially useful when sharing workflow definitions with a community. From the systems we examined, both *Pegasus* and*Makeflow* offer full support for local, cluster, grid and cloud execution. *Snakemake*, on the other hand, requires a shared file system and is not suitable for distributed sites like grids. All three systems that we examined allow for some form of job grouping, which means submitting a set of short running tasks as a single batch job.

8.3. Comparison of WMS Performance

For profiling runs with workflow management systems, we have seen that most of the workflow tasks have an almost equal runtime. In the cases where we do see significant

variations, it is not possible to determine a connection between job characteristics and execution time. To give an example: We see lower a execution time for *Snakemake* executions of I/O-intensive jobs from the *Montage* workflow. For more compute-intensive jobs like the mProject job however, the times are almost identical. This could lead to the hypothesis that *Snakemake* performs better with I/O-intensive jobs. But when considering the results of the *1000Genome* workflow runs, *Snakemake* offers the worst performance for the I/O-intensive individuals_merge task, which challenges this hypothesis. Consequently, the differences must have more complex reasons than a simple classification in CPU- or I/O-bound tasks.

With the results of our research, it is not possible to recommend a workflow management system for a certain type of workflow class with regard to the performance of the workflow jobs. However, for workflows with thousands of short running jobs, the overhead introduced by the batch system plays a more significant role [2]. This overhead is not reflected in the performance of the workflow tasks, but rather in the overall workflow wall time. Systems like *Pegasus*, which add multiple auxiliary jobs to transfer data between directories, add more overhead per job than systems with straightforward execution models, like *Snakemake*. The trade-off is that the auxiliary jobs allow for more flexibility when choosing an execution environment. *Snakemake* requires a shared file system, while *Pegasus* can be used for heterogeneous distributed execution sites like grids.

9. Conclusion

This chapter gives a brief summary of our results in section 9.1. Finally, section 9.2 offers an outlook on possible future work.

9.1. Summary

In this thesis, we have presented and compared three workflow management systems. For this purpose, we characterized three different workflows from different scientific domains regarding their resource requirements and utilization. While doing so, we focused on five input instances of varying complexity. The workflows were then used to evaluate the performance of the management systems and compare them to each other. With this approach, our research differentiates itself from related work that either examined only one system or used varying workflows and execution environments, which makes a comparison difficult. Our measurements have shown that the performance of workflow applications is not significantly influenced by the workflow management system in use. Instead, the overall wall time depends more on the dispatch overhead of the batch system or varying job grouping techniques in use.

Apart from performance, we have discussed the features and limitations of the *Pegasus*, *Snakemake* and *Makeflow* management systems to help researchers understand the capabilities and strengths of each system.

9.2. Future Work

In order to get a complete picture of the runtime behavior of scientific workflows, further research could focus on the overall wall time of workflow executions in an isolated environment, which offers more consistent wait times than a cluster shared with other users.

Furthermore, future work could utilize the task-level workflow characteristics that we collected to try and improve the process of job grouping that is performed by workflow management systems. A deeper understanding of the requirements can help to find a scheduling strategy that minimizes the overall execution time.

Acknowledgments

The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

This research made use of Montage. It is funded by the National Science Foundation under Grant Number ACI-1440620, and was previously funded by the National Aeronautics and Space Administration's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the California Institute of Technology.

This research used the Pegasus Workflow Management Software funded by the National Science Foundation under grant #1664162.

Bibliography

[1] B. P. Abbott et al. “LIGO: the Laser Interferometer Gravitational-Wave Observatory”. In: *Reports on Progress in Physics* 72.7 (June 2009), p. 076901. ISSN: 0034-4885. DOI: 10.1088/0034-4885/72/7/076901. URL: https://dx.doi.org/10.1088/0034-4885/72/7/076901 (visited on 01/12/2024).

[2] Michael Albrecht et al. “Makeflow: a portable abstraction for data intensive computing on clusters, clouds, and grids”. In: *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. SWEET ’12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 1–13. ISBN: 978-1-4503-1876-1. DOI: 10.1145/2443416.2443417. URL: https://dl.acm.org/doi/10.1145/2443416.2443417 (visited on 12/12/2023).

[3] Emir Bahsi, Emrah Ceyhan, and Tevfik Kosar. “Conditional Workflow Management: A Survey and Analysis”. In: *Scientific Programming* 15 (Jan. 1, 2007), pp. 283–297. DOI: 10.1155/2007/680291.

[4] Marc Bux and Ulf Leser. *Parallelization in Scientific Workflow Management Systems*. Mar. 28, 2013. DOI: 10.48550/arXiv.1303.7195. URL: http://arxiv.org/abs/1303.7195 (visited on 12/23/2023).

[5] *bwHPC Wiki*. URL: https://wiki.bwhpc.de/e/Main_Page (visited on 01/05/2024).

[6] Alexandru Calotoiu et al. “Fast Multi-parameter Performance Modeling”. In: *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 2016 IEEE International Conference on Cluster Computing (CLUSTER). ISSN: 2168-9253. Sept. 2016, pp. 172–181. DOI: 10.1109/CLUSTER.2016.57. URL: https://ieeexplore.ieee.org/abstract/document/7776507?casa_token=T3u9f6BzgpYAAAAA:eNhP2U5EooqlmOs-cSClCiEe1Y2mEjS3Lyx_xuJMDizjMzvRbJGVvnJKNyBivdG9c2f9YAbM (visited on 05/10/2024).

[7] Tainã Coleman et al. “WfBench: Automated Generation of Scientific Workflow Benchmarks”. In: *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). Nov. 2022, pp. 100–111. DOI: 10.1109/PMBS56514.2022.00014. URL: https://ieeexplore.ieee.org/abstract/document/10024036 (visited on 01/08/2024).

[8] Tainã Coleman et al. “WfCommons: A framework for enabling scientific workflow research and development”. In: *Future Generation Computer Systems* 128 (Mar. 1, 2022), pp. 16–27. ISSN: 0167-739X. DOI: 10.1016/j.future.2021.09.043. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21003897 (visited on 05/08/2024).

[9] Ewa Deelman et al. "Mapping Abstract Complex Workflows onto Grid Environments". In: *Journal of Grid Computing* 1.1 (Mar. 1, 2003), pp. 25–39. ISSN: 1572-9184. DOI: 10.1023/A:1024000426962. URL: https://doi.org/10.1023/A:1024000426962 (visited on 12/23/2023).

[10] Ewa Deelman et al. "Pegasus, a workflow management system for science automation". In: *Future Generation Computer Systems* 46 (May 1, 2015), pp. 17–35. ISSN: 0167-739X. DOI: 10.1016/j.future.2014.10.008. URL: https://www.sciencedirect.com/science/article/pii/S0167739X14002015 (visited on 01/08/2024).

[11] Ewa Deelman et al. "The Evolution of the Pegasus Workflow Management Software". In: *Computing in Science & Engineering* 21.4 (July 2019), pp. 22–36. ISSN: 1558-366X. DOI: 10.1109/MCSE.2019.2919690. URL: https://ieeexplore.ieee.org/abstract/document/8725518 (visited on 01/08/2024).

[12] Ewa Deelman et al. "The future of scientific workflows". In: *The International Journal of High Performance Computing Applications* 32.1 (Jan. 1, 2018), pp. 159–175. ISSN: 1094-3420. DOI: 10.1177/1094342017704893. URL: https://doi.org/10.1177/1094342017704893 (visited on 01/05/2024).

[13] Ewa Deelman et al. "Workflows and e-Science: An overview of workflow system features and capabilities". In: *Future Generation Computer Systems* 25.5 (May 1, 2009), pp. 528–540. ISSN: 0167-739X. DOI: 10.1016/j.future.2008.06.012. URL: https://www.sciencedirect.com/science/article/pii/S0167739X08000861 (visited on 12/23/2023).

[14] Rafael Ferreira da Silva et al. "A characterization of workflow management systems for extreme-scale applications". In: *Future Generation Computer Systems* 75 (Oct. 1, 2017), pp. 228–238. ISSN: 0167-739X. DOI: 10.1016/j.future.2017.02.026. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17302510 (visited on 11/27/2023).

[15] Ian Foster and Carl Kesselman, eds. *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Sept. 1998. 572 pp. ISBN: 978-1-55860-475-9.

[16] Dennis Gannon et al. "Workflows for e-Science". In: Jan. 1, 2007, pp. 1–8. ISBN: 978-1-84628-519-6.

[17] Andreas Gocht, Robert Schöne, and Jan Frenzel. "Advanced Python Performance Monitoring with Score-P". In: *Tools for High Performance Computing 2018 / 2019*. Ed. by Hartmut Mix et al. Cham: Springer International Publishing, 2021, pp. 261–270. ISBN: 978-3-030-66057-4. DOI: 10.1007/978-3-030-66057-4_14.

[18] *HTCondor*. URL: https://htcondor.org/ (visited on 05/15/2024).

[19] Gideon Juve et al. "Characterizing and profiling scientific workflows". In: *Future Generation Computer Systems*. Special Section: Recent Developments in High Performance Computing and Security 29.3 (Mar. 1, 2013), pp. 682–692. ISSN: 0167-739X. DOI: 10.1016/j.future.2012.08.015. URL: https://www.sciencedirect.com/science/article/pii/S0167739X12001732 (visited on 11/27/2023).

[20] Johannes Köster and Sven Rahmann. “Snakemake—a scalable bioinformatics workflow engine”. In: *Bioinformatics* 28.19 (Oct. 1, 2012), pp. 2520–2522. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts480. URL: https://doi.org/10.1093/bioinformatics/bts480 (visited on 01/12/2024).

[21] Dariusz Król et al. “Workflow Performance Profiles: Development and Analysis”. In: *Euro-Par 2016: Parallel Processing Workshops*. Ed. by Frédéric Desprez et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 108–120. ISBN: 978-3-319-58943-5. DOI: 10.1007/978-3-319-58943-5_9.

[22] Chee Sun Liew et al. “Scientific Workflows: Moving Across Paradigms”. In: *ACM Computing Surveys* 49.4 (2016), 66:1–66:39. ISSN: 0360-0300. DOI: 10.1145/3012429. URL: https://dl.acm.org/doi/10.1145/3012429 (visited on 01/05/2024).

[23] Ji Liu et al. “A Survey of Data-Intensive Scientific Workflow Management”. In: *Journal of Grid Computing* 13.4 (Dec. 1, 2015), pp. 457–493. ISSN: 1572-9184. DOI: 10.1007/s10723-015-9329-8. URL: https://doi.org/10.1007/s10723-015-9329-8 (visited on 12/23/2023).

[24] *Orcasound Project*. Orcasound. URL: https://www.orcasound.net/ (visited on 05/15/2024).

[25] *PAPI*. URL: https://icl.utk.edu/papi/.

[26] Lavanya Ramakrishnan. *A Survey of Distributed Workflow Characteristics and Resource Requirements*.

[27] Daniel Scheerer. *Analyzing Scientific Workflow Management Systems - Dataset*. Sept. 27, 2024. DOI: 10.5281/zenodo.13850444. URL: https://zenodo.org/records/13850444.

[28] *Score-P*. URL: https://www.vi-hps.org/projects/score-p/.

[29] Rafael Ferreira da Silva et al. “A Community Roadmap for Scientific Workflows Research and Development”. In: *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. Nov. 2021, pp. 81–90. DOI: 10.1109/WORKS54523.2021.00016. URL: http://arxiv.org/abs/2110.02168 (visited on 11/27/2023).

[30] *Slurm Workload Manager - Documentation*. URL: https://slurm.schedmd.com/documentation.html (visited on 01/11/2024).

[31] *Snakemake workflow catalog*. URL: https://snakemake.github.io/snakemake-workflow-catalog/ (visited on 01/12/2024).

[32] Jia Yu and Rajkumar Buyya. “A Taxonomy of Workflow Management Systems for Grid Computing”. In: *Journal of Grid Computing* 3.3 (Sept. 1, 2005), pp. 171–200. ISSN: 1572-9184. DOI: 10.1007/s10723-005-9010-8. URL: https://doi.org/10.1007/s10723-005-9010-8 (visited on 12/04/2023).

A. Appendix

A.1. Detailed Profiling Data

A.1.1. Montage

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	189.20	31.53	31.21	31.88	<0.01
1	598.05	31.48	29.22	32.41	<0.01
2	3113.47	31.45	26.08	32.86	<0.01
3	5111.70	31.55	26.61	33.61	<0.01
4	7731.90	31.56	26.33	33.92	<0.01

Table A.1.: **mProject** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	2.65	0.29	0.21	0.40	0.10
1	12.20	0.30	0.20	0.57	0.03
2	108.83	0.43	0.18	0.88	0.09
3	195.61	0.46	0.19	1.09	0.04
4	280.52	0.42	0.18	0.99	0.07

Table A.2.: **mDiffFit** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	0.22	-	-	-	0.13
1	0.65	-	-	-	0.16
2	5.57	-	-	-	0.03
3	10.17	-	-	-	0.03
4	15.37	-	-	-	0.06

Table A.3.: **mConcatFit** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	0.23	-	-	-	0.21
1	0.24	-	-	-	0.06
2	0.63	-	-	-	0.02
3	0.94	-	-	-	0.01
4	1.38	-	-	-	0.01

Table A.4.: **mBgModel** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	1.48	0.25	0.22	0.29	0.12
1	5.34	0.28	0.21	0.37	0.08
2	30.72	0.31	0.20	0.46	0.01
3	51.89	0.32	0.20	0.57	0.02
4	79.02	0.32	0.20	0.55	0.05

Table A.5.: **mBackground** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	0.14	-	-	-	0.08
1	0.42	-	-	-	0.38
2	2.95	-	-	-	0.13
3	4.73	-	-	-	0.09
4	7.23	-	-	-	0.09

Table A.6.: **mImgtbl** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	0.49	-	-	-	0.20
1	1.87	-	-	-	0.21
2	12.70	-	-	-	0.12
3	20.12	-	-	-	0.08
4	30.71	-	-	-	0.03

Table A.7.: **mAdd** job data with Pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	0.40	-	-	-	0.01
1	1.33	-	-	-	0.01
2	10.53	-	-	-	<0.01
3	18.90	-	-	-	<0.01
4	29.55	-	-	-	<0.01

Table A.8.: **mViewer** job data with Pegasus

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	189.37	31.56	31.28	31.83	<0.01	1.13×10^9	<0.01
1	597.44	31.44	29.26	31.97	<0.01	1.13×10^9	<0.01
2	3119.39	31.51	26.08	32.86	<0.01	1.13×10^9	<0.01
3	5114.30	31.57	26.76	33.00	<0.01	1.13×10^9	<0.01
4	7747.75	31.62	26.51	33.68	<0.01	1.12×10^9	<0.01

Table A.9.: **mProject** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	2.34	0.26	0.22	0.35	0.09	6.20×10^6	0.08
1	10.28	0.26	0.18	0.42	0.07	6.76×10^6	0.04
2	66.28	0.26	0.18	0.81	0.05	6.66×10^6	0.03
3	117.52	0.27	0.18	0.74	0.05	6.38×10^6	0.04
4	229.31	0.35	0.19	0.90	0.14	5.37×10^6	0.12

Table A.10.: **mDiffFit** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.16	-	-	-	0.08	740.42	0.09
1	0.46	-	-	-	0.21	983.91	0.19
2	4.17	-	-	-	0.26	695.85	0.30
3	5.65	-	-	-	0.22	845.76	0.18
4	14.39	-	-	-	0.06	497.61	0.06

Table A.11.: **mConcatFit** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.18	-	-	-	0.05	7.80×10^7	0.05
1	0.23	-	-	-	0.06	2.33×10^8	0.05
2	0.62	-	-	-	0.06	7.89×10^8	0.06
3	0.96	-	-	-	0.03	9.10×10^8	0.03
4	1.39	-	-	-	0.01	9.95×10^8	0.01

Table A.12.: **mBgModel** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	1.29	0.21	0.21	0.22	0.01	2.18×10^7	0.01
1	4.05	0.21	0.20	0.23	0.01	2.17×10^7	0.01
2	21.16	0.21	0.20	0.24	<0.01	2.17×10^7	<0.01
3	35.06	0.22	0.20	0.29	0.01	2.16×10^7	0.01
4	62.89	0.26	0.20	0.40	0.01	1.87×10^7	0.01

Table A.13.: **mBackground** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.12	-	-	-	0.11	8.94×10^4	0.11
1	0.26	-	-	-	0.48	1.51×10^5	0.32
2	1.76	-	-	-	0.63	1.34×10^5	0.51
3	2.07	-	-	-	0.60	1.76×10^5	0.47
4	6.07	-	-	-	0.03	7.18×10^4	0.03

Table A.14.: **mImgtbl** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.37	-	-	-	0.24	1.04×10^7	0.19
1	1.28	-	-	-	0.30	1.37×10^7	0.32
2	9.43	-	-	-	0.25	1.53×10^7	0.24
3	14.22	-	-	-	0.25	1.77×10^7	0.21
4	28.18	-	-	-	0.02	1.33×10^7	0.02

Table A.15.: **mAdd** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.27	-	-	-	0.03	6.02×10^7	0.03
1	1.44	-	-	-	0.03	1.66×10^7	0.03
2	13.07	-	-	-	0.02	7.77×10^6	0.02
3	22.79	-	-	-	<0.01	7.44×10^6	<0.01
4	35.63	-	-	-	<0.01	7.22×10^6	<0.01

Table A.16.: **mViewer** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	189.57	31.59	31.22	32.24	0.01	1.13×10^9	0.01
1	597.37	31.44	29.24	32.27	<0.01	1.13×10^9	<0.01
2	3112.53	31.44	26.03	32.62	<0.01	1.13×10^9	<0.01
3	5114.83	31.57	26.71	33.16	<0.01	1.13×10^9	<0.01
4	7731.94	31.56	26.34	33.15	<0.01	1.13×10^9	<0.01

Table A.17.: **mProject** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	2.60	0.29	0.24	0.36	0.09	5.59×10^6	0.06
1	12.32	0.31	0.22	0.47	0.03	5.61×10^6	0.03
2	102.37	0.41	0.21	0.99	0.03	4.65×10^6	0.02
3	182.84	0.43	0.20	1.15	0.05	4.45×10^6	0.04
4	282.96	0.43	0.19	1.53	0.07	4.49×10^6	0.07

Table A.18.: **mDiffFit** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.22	-	-	-	0.23	548.08	0.21
1	0.65	-	-	-	0.10	681.62	0.11
2	5.59	-	-	-	0.09	490.39	0.10
3	9.41	-	-	-	0.05	492.51	0.05
4	15.29	-	-	-	0.11	471.46	0.10

Table A.19.: **mConcatFit** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.20	-	-	-	0.15	7.13×10^7	0.14
1	0.25	-	-	-	0.10	2.16×10^8	0.09
2	0.61	-	-	-	0.02	8.01×10^8	0.02
3	0.96	-	-	-	0.01	9.12×10^8	0.01
4	1.39	-	-	-	0.01	9.92×10^8	0.01

Table A.20.: **mBgModel** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	1.36	0.23	0.21	0.24	0.02	2.06×10^7	0.02
1	4.29	0.23	0.21	0.26	0.02	2.06×10^6	0.02
2	27.81	0.28	0.20	0.42	0.08	1.71×10^7	0.08
3	50.01	0.31	0.20	0.53	0.06	1.56×10^7	0.06
4	77.07	0.31	0.20	0.49	0.02	1.52×10^7	0.03

Table A.21.: **mBackground** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.20	-	-	-	0.66	6.82×10^4	0.35
1	0.28	-	-	-	0.15	1.25×10^5	0.14
2	2.82	-	-	-	0.34	7.08×10^4	0.44
3	4.44	-	-	-	0.07	6.52×10^4	0.07
4	7.41	-	-	-	0.03	5.88×10^4	0.03

Table A.22.: **mImgtbl** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.32	-	-	-	0.15	1.18×10^7	0.13
1	1.13	-	-	-	0.30	1.54×10^7	0.27
2	12.06	-	-	-	0.16	1.17×10^7	0.20
3	18.34	-	-	-	0.05	1.32×10^7	0.05
4	30.01	-	-	-	0.06	1.26×10^7	0.06

Table A.23.: **mAdd** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	0.41	-	-	-	0.04	4.00×10^7	0.04
1	1.33	-	-	-	0.04	1.79×10^7	0.04
2	10.61	-	-	-	<0.01	9.58×10^6	<0.01
3	18.90	-	-	-	0.01	8.98×10^6	0.01
4	29.67	-	-	-	0.01	8.67×10^6	0.01

Table A.24.: **mViewer** job data with makeflow

A.1.2. 1000Genome

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	87 194.70	43 597.35	43 421.35	43 773.35	<0.01
1	86 095.58	21 523.89	21 254.03	21 763.72	<0.01
2	85 319.39	8531.94	8418.35	8578.85	<0.01
3	85 024.73	4251.24	4160.20	4275.24	<0.01
4	84 978.91	2124.47	2043.91	2141.54	<0.01

Table A.25.: **individuals** job data with pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	285.83	-	-	-	0.01
1	313.18	-	-	-	0.01
2	382.41	-	-	-	0.05
3	471.44	-	-	-	0.07
4	586.54	-	-	-	0.03

Table A.26.: **individuals_merge** job data with pegasus

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	87 449.89	43 724.95	43 609.20	43 840.70	<0.01	2127.51	<0.01
1	86 128.50	21 532.13	21 232.27	21 767.01	<0.01	2168.87	<0.01
2	85 392.58	8539.26	8417.74	8607.33	<0.01	2214.85	<0.01
3	85 115.27	4255.76	4177.82	4282.07	<0.01	2267.48	<0.01
4	85 111.25	2127.78	2042.77	2140.20	<0.01	2357.47	<0.01

Table A.27.: **individuals** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	296.55	-	-	-	0.02	1837.60	0.02
1	307.51	-	-	-	0.03	3149.88	0.02
2	370.61	-	-	-	0.03	6043.58	0.03
3	482.05	-	-	-	0.07	9071.36	0.07
4	649.93	-	-	-	0.05	1.32×10^4	0.05

Table A.28.: **individuals_merge** job data with snakemake

		Time (s)					FLOPS	
Job	Instance	Total	Avg	Min	Max	CoV	Avg	CoV
sifting	3	3.98	-	-	-	0.02	6.79×10^4	0.02
mutation_overlap	0	183.25	26.18	16.67	56.02	0.04	9.38×10^5	0.03
frequency	3	1437.36	205.34	173.32	260.45	<0.01	9.05×10^5	<0.01

Table A.29.: **other** job data

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	86 991.98	43 495.99	43 429.83	43 562.15	<0.01	2138.72	<0.01
1	86 035.68	21 508.92	21 234.76	21 752.27	<0.01	2171.18	<0.01
2	85 273.12	8527.31	8415.84	8590.58	<0.01	2217.96	<0.01
3	85 065.07	4253.25	4168.02	4277.51	<0.01	2268.90	<0.01
4	85 060.58	2126.51	2045.41	2139.96	<0.01	2358.87	<0.01

Table A.30.: **individuals** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	288.73	-	-	-	0.02	1887.49	0.02
1	304.37	-	-	-	0.01	3181.00	0.01
2	355.30	-	-	-	0.01	6299.78	0.01
3	447.10	-	-	-	0.02	9744.52	0.02
4	618.99	-	-	-	0.07	1.39×10^4	0.07

Table A.31.: **individuals_merge** job data with makeflow

		Time (s)					FLOPS	
Job	Instance	Total	Avg	Min	Max	CoV	Avg	CoV
sifting	2	4.72	-	-	-	0.05	5.74×10^4	0.05
mutation_overlap	3	245.84	26.18	16.67	56.02	0.04	9.38×10^5	0.03
frequency	3	1437.36	205.34	173.32	260.45	<0.01	9.05×10^5	<0.01

Table A.32.: **other** job data

A.1.3. Orcasound

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	101.16	-	-	-	0.13
1	217.47	108.74	107.74	109.73	0.01
2	434.90	108.73	103.58	113.69	0.02
3	910.80	113.85	108.75	118.05	0.06
4	1811.50	113.22	107.68	118.55	0.04

Table A.33.: **convert2wav** job data with pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	684.71	-	-	-	0.01
1	1396.80	698.40	689.17	707.64	0.03
2	2778.47	694.62	679.16	723.32	0.01
3	5560.20	695.03	684.50	711.67	0.01
4	11 156.45	697.28	673.47	731.52	0.01

Table A.34.: **convert2spectrogram** job data with pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
0	430.64	-	-	-	0.11
1	783.21	391.60	364.50	418.71	0.06
2	1610.04	402.51	364.37	425.26	0.05
3	3352.33	419.04	407.72	457.68	0.06
4	6491.08	405.69	361.37	447.75	0.03

Table A.35.: **inference** job data with pegasus

	Time (s)				
Instance	Total	Avg	Min	Max	Var
1	0.21	-	-	-	0.06
2	0.36	-	-	-	0.13
3	0.59	-	-	-	0.08
4	1.05	-	-	-	0.06

Table A.36.: **merge** job data with pegasus

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	108.66	-	-	-	0.17	38.12	0.22
1	199.77	99.89	96.88	102.89	0.01	40.24	0.01
2	400.77	100.19	90.73	107.78	0.08	40.49	0.08
3	854.88	106.86	100.81	110.22	0.05	37.69	0.05
4	1726.93	107.93	98.28	114.37	0.05	37.34	0.05

Table A.37.: **convert2wav** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	695.89	-	-	-	0.04	3.21×10^8	0.04
1	1386.65	693.32	690.52	696.13	0.01	3.22×10^8	0.01
2	2831.78	707.94	690.03	722.41	0.03	3.16×10^8	0.03
3	5555.06	694.38	677.12	727.81	<0.01	3.22×10^8	<0.01
4	11 237.08	702.32	676.76	737.01	<0.01	3.17×10^8	<0.01

Table A.38.: **convert2spectrogram** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	409.89	-	-	-	0.07	1.01×10^8	0.06
1	894.75	447.38	439.42	455.34	0.03	9.27×10^7	0.03
2	1534.18	383.55	357.43	416.64	0.06	1.09×10^8	0.05
3	3155.15	394.39	355.18	450.28	0.02	1.06×10^8	0.02
4	6024.46	376.53	336.08	421.07	0.01	1.10×10^8	0.01

Table A.39.: **inference** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
1	0.19	-	-	-	0.09	8.41×10^5	0.10
2	0.31	-	-	-	0.08	1.03×10^6	0.08
3	0.57	-	-	-	0.09	1.10×10^6	0.09
4	1.01	-	-	-	0.07	1.22×10^6	0.06

Table A.40.: **merge** job data with snakemake

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	82.57	82.57	82.57	82.57	0.07	48.87	0.07
1	166.31	83.15	82.01	84.30	0.05	48.47	0.05
2	373.93	93.48	87.39	98.65	0.07	43.28	0.06
3	732.35	91.54	85.87	102.34	0.05	44.19	0.06
4	1747.56	109.22	94.23	116.99	0.04	37.04	0.04

Table A.41.: **convert2wav** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	691.63	-	-	-	0.01	3.23×10^8	0.01
1	1375.19	687.59	685.81	689.38	0.01	3.25×10^8	0.01
2	2731.32	682.83	676.17	688.97	0.01	3.28×10^8	0.01
3	5475.73	684.47	677.57	695.12	0.01	3.27×10^8	0.01
4	11 014.18	688.39	674.40	706.66	0.01	3.24×10^8	0.01

Table A.42.: **convert2spectrogram** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
0	411.45	-	-	-	0.05	1.01×10^8	0.05
1	794.91	397.46	387.66	407.25	0.03	1.04×10^8	0.03
2	1498.47	374.62	364.59	384.72	0.01	1.11×10^8	0.01
3	3215.14	401.89	390.68	425.40	0.06	1.04×10^8	0.06
4	6302.33	393.90	362.91	427.14	0.04	1.05×10^8	0.04

Table A.43.: **inference** job data with makeflow

	Time (s)					FLOPS	
Instance	Total	Avg	Min	Max	Var	Avg	Var
1	0.19	-	-	-	0.11	8.56×10^5	0.10
2	0.29	-	-	-	0.22	1.11×10^6	0.17
3	0.49	-	-	-	0.20	1.31×10^6	0.16
4	0.89	-	-	-	0.07	1.40×10^6	0.08

Table A.44.: **merge** job data with makeflow