Accurate Performance Predictions with Component-based Models of Data Streaming Applications

Dominik Werle(✉)[0000−0002−2430−2578], Stephan Seifermann[0000−0002−9727−0407], and Anne Koziolek[0000−0002−1593−3394]

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
`dominik.werle@kit.edu`, `stephan.seifermann@kit.edu`, `koziolek@kit.edu`

Abstract. Data streaming applications are an important class of data-intensive systems and performance is an essential quality of such systems. Current component-based performance prediction approaches are not sufficient for modeling and predicting the performance of those systems, because the models require elaborate manual engineering to approximate the behavior of data streaming applications that include stateful asynchronous operations, such as windowing operations, and because the simulations for these models do not support the metrics that are specific to data streaming applications. In this paper, we present a modeling language, a simulation and a case-study-based evaluation of the prediction accuracy of an approach for modeling systems that contain stateful asynchronous operations. Our approach directly represents these operations and simulates their behavior. We compare measurements of relevant performance metrics to performance simulation results for a system that processes smart meter readings. To assess the prediction accuracy of our model, we vary both the configuration of the streaming application, such as window sizes, as well as the characteristics of the input data, i.e., the number of smart meters. Our evaluation shows that our model yields prediction results that are competitive with a state-of-the-art baseline model without incurring the additional manual engineering overhead.

Keywords: software performance · model-driven software engineering · computer aided software engineering · software architecture · data streaming · big data applications

1 Introduction

Systems that process large amounts of data from varied sources are becoming a more and more important class of software systems in recent years, for which numerous frameworks, middlewares and overall implementation techniques exist. Reasons for the increased demand for this type of systems are the increasing amount of data sources from which data is collected and a steady increase in the number of models and methods for data analysis. One manifestation of this is the ongoing growth in popularity for machine learning models.

However, it is challenging for software engineers to build systems that process large amounts of data [8]. Part of the challenge is planning how the system will perform for future workloads that are either larger in scale or that have different characteristics that influence how data is grouped and processed. One approach to systematically analyze the quality of a software system are models, which allow exploring different implementation alternatives without fully implementing and running them. They allow analyzing the performance of the system under changing workloads without investing the resources needed for extensive load tests. Recent research to reduce the cost of automatically building and maintaining models of software systems for which asynchronous communication is a major performance influence factor [17] has made the use of models for these software systems a relevant alternative in comparison to building and load-testing implementations.

In this paper, we examine whether component-based performance models that include stateful operations can be used to predict the performance of data-intensive software systems as accurately as the state of the art while using models that are better aligned with the systems. We call these operations *stateful asynchronous operations (SAOs).* This question is relevant because our previous research [19] indicates that some of the timing effects present in this type of system cannot be suitably predicted with stateless models and require approximations and workarounds. An example of a stateful operation is when data is collected and emitted as a group after a specified duration has passed or after a specified number of elements has arrived. Such stateful operations are common in data stream processing applications, for example as sliding windows that are created on data streams [1]. We use a case study to evaluate the benefit of models extended with SAOs. We extend our previous research in performance modeling for data-streaming applications [19] with a generalization of the model and simulation and with an evaluation of the resulting approach.

Derived from the presented problem, we address the following research question: RQ_1: Does explicitly modeling stateful asynchronous operations allow modelers to create architecture-level performance models that are better aligned with the system architecture without reducing the prediction accuracy of the models?

To answer RQ_1, we model different configurations of a case study system both with our approach (*model with stateful asynchronous operations (MSAO)* hereafter), which includes SAOs, and with systematically created baseline models using a state-of-the-art modeling approach (*baseline (BL)* hereafter), which does not include SAOs. We discuss the process that these models are derived by and how the models align with changes in configuration and workload. We furthermore evaluate the accuracy of these models in comparison with measurements on an implementation of the case study system. The metric that we consider in this evaluation is the data age after analysis, which is the time from the creation of a data item to the point of time it is included in an analysis result. We focus on the metric data age because a) it is suited better for asynchronous processing than the more commonly supported metric *user response time* because one call to the system interface providing one data element can imply multiple, delayed

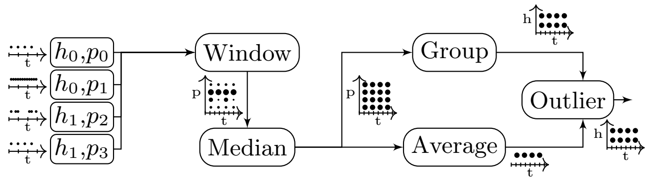

Fig. 1. Illustration of the case study system (Source: [18])

calls within the system, and b) it is not as well supported with current modeling approaches. Our evaluation shows that MSAO yields prediction results that are competitive with BL without incurring the additional manual engineering overhead. Our evaluation investigates the usefulness of MSAO in comparison with the state of the art and we do not intend to formally prove that current approaches theoretically are not able to represent streaming behavior. Thus, we create a baseline model that is as accurate as possible and explain the behavior that the analyst has to manually represent in the baseline by calculating stochastic descriptions of the behavior by hand. In contrast, our approach provides explicit modeling constructs for the behavior. Measuring the modeling effort in a statistically sound way requires a controlled experiment, which is out of scope for this article. Thus, we use a discussion for presenting the reduced manual overhead as clearly as possible. We plan to evaluate increases in the accuracy of the prediction in future work for scenarios where the performance of the system heavily depends on the underlying technical realization of the middleware in cases where delays caused by scheduling on active resources dominate the performance.

We present the following contributions: C_1: A simulation approach for system models that incorporate stateful asynchronous operations. C_2: A case-study based evaluation of our modeling and simulation approach. The data and models used in the experiment are available in the companion data set [20].

2 Running Example

For our case study, we implemented and instrumented a case study system that we described and published in previous work [19,18]. It is based on the grand challenge of the DEBS Conference on Distributed and Event-based Systems 2014 [10,9]. The system creates sliding windows of smart meter readings to calculate outlier values for each house and time window. The outlier value of a house for a window is the percentage of smart plugs in the house that have a median reading value for the window that is above the overall average median reading value of all plugs for this window. We chose this system since its complexity is manageable for implementation and measuring with different system setups. The case study system is illustrated in Figure 1. It is structured in five components: *Window* creates groups of data elements via sliding windows. The groups are partitioned according to plugs. The component emits one collection of readings for each smart plug p and time window. *Median* takes readings, calculates a median of the values and emits this median value. Therefore, the component emits one median per plug for each time window. *Group* collects the medians and groups them by the house h the plugs belong to. Thus, it emits one collection of medians for each house and window. *Average* calculates one overall average of all medians for each window and emits it. *Outlier* joins the medians that have been grouped per house together with the overall average of all medians. It operates once per house and time window to calculate one outlier per house per time window.

For our running example, the question that a software designer wants to answer during the design of the systems are about the configuration of the operations and for the behavior in different load scenarios. For example, they want to know how a change in the windowing operator, e.g., by increasing the spacing between windows, impacts the performance of the system, or how the system will perform when the amount of smart meters is doubled.

3 Related Work

We see two groups of works related to the approach presented in this paper: The first group provides modeling approaches for data-intensive applications (DIAs) in the architectural design phase. The second group supports the implementation of DIAs without a particular focus on the architectural design phase.

In the first group focusing on the architectural design phase, the approaches can be divided into approaches for streaming and approaches for batch processing. The work of Kroß et al. [11,12] targets stream processing. They extract Palladio performance models [14] for the Big Data frameworks Apache Spark and Hadoop, which they use for predicting the performance of DIAs. In contrast to our approach, Kroß et al. do not model stateful operations but derive their effect based on impact factors, such as the number of partitions in data streams, which architects have to identify first. The DICE project [5] provides methods for modeling and simulating [7] Big Data systems. The models distinguish the platform, technology and deployment and various combinations of these. For instance, the model supports Apache Storm topologies using different types of bolts. The simulation of the systems is based on Petri nets. However, the models do not consider stateful operations, to the best of our knowledge. Maddodi et al. [13] analyze the performance of event-sourcing applications by Layered Queuing Networks (LQNs). In contrast to our approach, they only support aggregating multiple calls but do not support other interactions of calls including windowing or joining data. There are various other approach for batch processing, which focus on different aspects of the analysed systems. Castiglione et al. [6] focus on predicting metrics relevant for operating highly concurrent applications in cloud infrastructure such as performance, number of virtual machines or energy efficiency. Aliabadi et al. [2] focus on predicting the performance of batch applications in different Big Data analysis frameworks using Stochastic Activity Networks. These approaches for batch processing do not support the metrics required for stream processing, which we focus on.

The second group of approaches does not focus on the architectural design phase, resulting in models with inappropriate abstractions for this phase. Sachs [16] suggests a model-based approach for predicting the performance of event-based systems based on Queueing Petri Nets. The presented Petri Net patterns support state-based behavior, such as time windows. In contrast to our approach, Sachs does not consider the decomposition of systems, which is essential for describing software architectures. The approach furthermore does not derive characteristics of resulting data elements and groups of elements after processing which can be measured or used as input for subsequent behavior. Wu et al. [21] provide an approach to formulate information needs in form of queries on event streams and provide instructions on implementing the queries with good performance, however not providing means of abstracting from the actual query plan, e.g., via stochastical descriptions of the used data or the load on the system.

4 Problem Analysis

In this section, we structure the influence factors for the performance of data-streaming applications in more detail. The benefit of this structuring is that we can align our modeling concepts (section 5) and our evaluation (section 7) with the identified factors. We then introduce required capabilities of the modeling language and simulation to represent stateful asynchronous operations in an architecture-level performance model. The collected factors are derived from current state-of-the-art in component-oriented performance modeling [14] and from work in defining operations that are used in data streaming applications (Apache Beam, Apache Flink, Dataflow Model [1]). Overall, our analysis is aligned with our previous work in component-oriented modeling of the performance of data streaming applications [19]. However, we generalize the state-of-the-art to allow a model that is as flexible as possible regarding the type of operations that can be represented. This is achieved by allowing the person that creates stateful asynchronous operations to implement the behavior of an operation as code that is included in the simulation.

4.1 Types of Delays

First, we identify different factors that influence the time it takes for a data item from its creation to its inclusion in an analysis result.

Active Resource Delays are introduced because scheduled resources such as CPUs are contested by different processes. Requests are scheduled and processing time depends on the ressource's characteristics.

Lock Delays occur when parts of the system can only be entered by one process concurrently, resulting in wait times for other processes.

Collective Operation Delays occur when the application waits for some time or event to further process a data. This is for example the case, when sliding windows of incoming data are created and emitted periodically.

4.2 Required Capabilities of the Modeling Language and Simulation

We identify the following required capabilities of the modeling language and simulation to represent stateful asynchronous operations and motivate each of the capabilities with a type of system behavior or with a requirement that comes from the way systems are modeled.

Asynchronous forks and joins of calls. The modeling language has to be able to express calls that result in multiple, parallel calls and a join of multiple, parallel calls. Forked calls do not have to be joined but can result in different computations in the system and can be included in different types of calculations and results.

Metrics for collective operations need to allow tracing a data item across different calls, e.g., a relevant metric can be the age of a data item since it has entered the system. If data is passed between asynchronous calls, this implies that the age has to be traced back to the entry point of the system. A data item does not necessarily have a single origin or a single time of processing, e.g., when a group of elements is calculated and further processed.

Characteristics of data must be representable by the modeler of the system, if they are performance-relevant. Characteristics means that a value is associated with the data. The value is not necessarily the payload of the data itself, but an abstract representation of the information that performance-relevant, i.e., it influences the observable timing of the system in some way. For example, the number of elements that have to be processed in a group of data elements or the characterization of the data that is used for filtering has an observable effect on the timing. This is in line with the way current component-oriented performance models represent characteristics [14].

References to characteristics of data must be possible in the model, i.e., the system can be expressed in a way that depends on characteristics of data. An example is filtering data elements by a characteristic, e.g., discarding elements that are larger that some threshold.

Reference to internal component state. Components need to be able to have state *during execution* which can influence the observable behavior for incoming calls. An example for why internal component state is required is implementing a maximum capacity of a queuing component that may block processes with new data, when the capacity is surpassed. This is only possible, if the component actively manages a queue of data elements.

Collective operations that derive characteristics. If the behavior can depend on the characteristics of data, we need to accurately compute the characteristics of data derived in the system. An example is when a stateful asynchronous operation partitions data based on its characteristics. Subsequently, the number of elements in each partition can be relevant for timing of further processing.

Stochastical behavior description is needed to define how operators and components behave based on stochastic distributions, for example taken from measurements. Stochastics are beneficial when we do not want invest the resources to model in detail the characteristics of data that is relevant for the behavior of the system. An example is a filtering component for which we provide a probability for a data item to be filtered instead of modeling the data characteristic that leads to filtering.

5 Modeling Concepts

In this section, we present and discuss the different modeling concepts for stateful asynchronous operations, which our approach introduces. While our approach addresses a similar problem as our previous work [19], we provide a more general and more flexible modeling and simulation approach. We structure the required concepts in three groups: a) stateful operations, b) (asynchronous) data-oriented call flow, c) metrics.

Stateful operations. In principle, a stateful operation can implement any type of behavior. Note that if the expressiveness is not constrained, we do not claim static analysability of the models. This means that we trade unrestricted expressiveness of the state and analysis by simulation for a reduction in the static analysability. To reduce the overall complexity, we can introduce some reusability for parts of the specification of stateful asynchronous operations. One possibility is to do this as a reduced modeling language for operators as introduced in our previous work [19]. While the previously described language introduces a predefined structure for queues and operations between them, we rely on black-box operations that can be implemented as Java code as part of the simulation. However, we allow similar composability by code reuse, e.g., by inheritance or composition of classes. To reduce the complexity of implementing the used SAOs in the simulation, we provide some common patterns.

Each stateful asynchronous operation has an internal time which it is certain has already passed. This concept is called watermarks or event time/processing time in literature [1] and in state-of-practice implementations (Apache Beam[1], Apache Flink[2]). Depending on the configuration, this time can be advanced either a) periodically to a reference clock, or b) when a new data item with a timestamp arrives. When the clock is advanced, the stateful asynchronous operation can decide whether it should process the current data. This is for example the case, when the new advanced time surpasses the end of a window that can in turn be created and emitted.

(Asynchronous) Data-oriented Call Flow. Calls through the system can be triggered in different ways. Periodic calls that are triggered according to operator properties, e.g., because windows are emitted periodically. Calls that are triggered because some predicate over previous calls is true, e.g., because the maximum distance to a previous call is surpassed. Furthermore, we can differentiate active data retrieval operations, where a call retrieves data from a data source and might block if no data is available, and pushing calls that are triggered as new processes in the system because data becomes available at a data source. Apart from the initial spawning of a process, calls may need to access

[1] `https://beam.apache.org/documentation/basics/#watermarks`
[2] `https://nightlies.apache.org/flink/flink-docs-stable/`

shared resources that represent a data structure, e.g., for retrieving or storing the actual data, e.g., in some type of shared concurrent data structure.

Metrics. When we focus on data that flows through the system, the simulation and model have to provide metrics that are data-oriented. Instead of investigating users that get a response from a business system, resulting in a response time metric, the model has to provide metrics that represent the timeliness of data elements at different points in the system. Particularly, for asynchronous operations (data elements can be forked and joined or otherwise replicated), we need a way to measure the time from the inception of the data (i.e., when it is created, e.g., in a sensor or in a clickstream) to points in the execution, for example the point of time when a data element is incorporated in an analysis (i.e., a specific computation), or when it becomes part of the active behavior of a stateful asynchronous operation (for example because it is discarded). Utilizing these metrics, the system designer can evaluate whether timeliness and quality requirements at different points in the system are met by different system designs and the configuration of the processing system while also considering the tradeoff with the characteristics of the processed data. For example, lowering the number of elements that are waited for, grouped, and processed can reduce the data age at the point of analysis. However, including less elements in the processing might provide less stable results, e.g., for a smoothed average.

6 Implementation

Our approach is an extension to an existing architecture description language designed for software quality predictions, the Palladio Component Model (PCM) [14], and an extension to a discrete event simulation for this modeling language, the SimuLizar [4] simulator. The components that are used in PCM to describe entities that provide or consume services and are deployed as a unit are called *basic components*. In contrast, we will call our components *data channels*. Basic components provide and require services and have a service effect description (SEFF) for each provided service. The SEFF describes the behavior of a call of a provided service in terms of resource demands and calls to other services. The behavior described in a SEFF (and thus a basic component) can only be triggered by an incoming call, either by a user or another component. In contrast, our approach includes two new types of role that a stateful asynchronous operations can play: data source role and data sink role. A data source role means that a component can act as a data source for a specific type of data item, i.e., it can emit this type of data. A data sink role means that a component can accept this type of data. In contrast to SEFFs, the behavior description for a stateful asynchronous operations has access to the aforementioned types of state (see subsection 4.2) and can actively emit data that can result in additional processing threads in the system. Our approach does not restrict the passing of data to push behavior (actively triggering processing in another part of the system) or pull behavior (consumers decide when they consume data from a data source) but allows to model both. To support simulating asynchronously processed data and grouping of different data in the system, we also make the chain of calls that a data item has passed through available in the simulation.

7 Evaluation

The aim of our evaluation is to show that our approach (MSAO) performs as well as a baseline state-of-the-art modeling approach that does not incorporate stateful asynchronous operations (BL). In this section, we will first define our evaluation goal and derive evaluation questions and metrics from it. We then describe the state-of-the-art baseline model without stateful asynchronous operations, and the model in our approach with stateful asynchronous operations. We then discuss how both models are calibrated. Last, we describe the space of scenarios that the models are not calibrated for and for which we predict the performance of the system.

7.1 Goals, Questions, Metrics

Our evaluation is structured along the Goal-Question-Metric (GQM) approach [3]. The evaluation goals are: G_1: Evaluate the accuracy of MSAO in comparison to BL for data streaming applications. G_2: Evaluate the modeling overhead for MSAO in comparison to BL for data streaming applications.

Therefore, the evaluation concerns systems using our modeling approach (contribution C_1). We derive the following evaluation questions from G_1: $EQ_{1.1}$: How accurate is the prediction of the distribution of data age at the time it is processed using BL? $EQ_{1.2}$: How accurate is the prediction of the distribution of data age at the time it is processed using MSAO?

We address these questions with the following metrics: M_1: Relative error of predicted average data age. M_2: Wasserstein metric between measured and predicted distribution of data age. We evaluate metrics M_1 and M_2 for all scenarios in our test set and also provide statistical data about the metrics (mean and maximum). Thus, $EQ_{1.1}$ and $EQ_{1.2}$ are evaluated quantitatively.

We derive the following evaluation question from G_2: $EQ_{2.1}$: What is the increase in complexity of modeling the system from BL to MSAO? We address this question by elaborating on the overhead of creating BL in subsection 7.4. Thus, $EQ_{2.1}$ is evaluated qualitatively via discussion.

7.2 Evaluation Design

To evaluate G_1 and G_2, we create both BL and MSAO for a case-study system. We calibrate both models using the same calibration measurements from our real implementation (training set). However, the active resource delays are negligible in comparison with the collective operation delays in the scenarios that we observed. For our test set, we choose different scenarios that change aspects of the input data or configuration of the system. For each of the scenarios, we again measure the behavior of the real implementation. We then compare the results of each of the models with the real measurements using M_1 and M_2 for the measured data age.

7.3 Experiment Setup

We have measured all combinations of the following configuration dimensions and values, leading to a total of $3 \cdot 10 = 30$ scenarios: i) window size/shift: 5/5, 10/10, 20/20, ii) workload: 1s1p1h, 1s2p2h, 1s4p4h, 1s8p8h, 1s10p10h, debs-2, debs-4, debs-all-2, debs-all-4, debs-all-6 The 5 workloads with identifiers of the form *1s2p2h* describe artificial workloads that have a fixed amount of houses and plugs. Each plug sends a power consumption measurement with a fixed frequency. In this case, the identifier 1s2p2h means that the frequency is 1 second, there are 2 plugs per house and 2 houses, leading to a total of 4 messages per second. We use these as training for our calibration. For our test set, we use 5 reduced excerpts of the original DEBS data set which are filtered variants of the first five minutes. Here, *debs-n* is a subset which only includes the first n houses. *debs-all-n* is a subset that only includes the first n plugs of the first n households of the first n houses. All of our measurements result in a steady state, i.e., they do not lead to a congestion behavior that overloads the system indefinitely.

The measurements are done on a machine with an AMD Ryzen 9 3900X 12-Core Processor running Windows 10, Version 21H1. All of the measurements are executed in the Windows Subsystem for Linux (WSL) running Ubuntu 20.04.3 LTS. The application is executed as a Java program that runs inside one instance of the Java Virtual Machine (JVM). The Java version we use is OpenJDK 11.0.11. Each component is run as one thread inside the JVM. We pin the `java` executable to the first processor core using `taskset -pc 0`. We further allow limiting the CPU usage of the process using `cpulimit -b -m -l <percentage>`. For our experiments, we use a limit of 100 %, i.e., to a maximum of one CPU.

7.4 Models

In this section, we describe how both BL as well as MSAO are created and how the models are derived for the different modeling scenarios. The goal of this section is to answer $EQ_{2.1}$. The proposed benefit of creating models with first-class entities for SAOs is twofold: i) architects do not need to create workarounds with current techniques that only capture a specific configuration of the system, ii) they can directly measure the metrics of interest instead of relying on proxy metrics. In the following, we will illustrate these workarounds for BL.

We implemented an automated tool that uses measurements from the training set for calibrating the resource demands that occur in the system. Our tool creates models for both BL and MSAO for each scenario from the test set, automatically executes the simulation and stores the simulation results. We define which of the measurements belong to the test set and which belong to the training set and automatically derive M_1 and M_2.

Baseline Model. Figure 2 illustrates our generated baseline model. The model consists of four components that represent the different parts of the system that

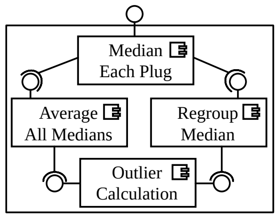

Fig. 2. Baseline Model

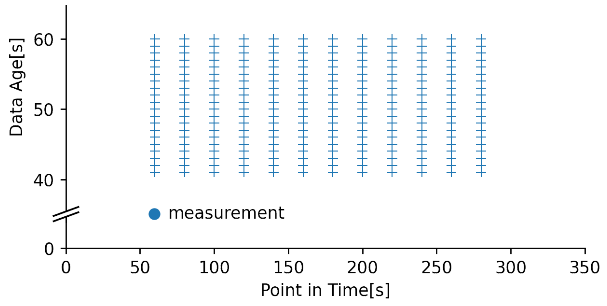

Fig. 3. Measurement for artificial scenario 1s4p4h, window size/shift 20/20.

incur active resource delays. The usage model of the baseline model reproduces the windowing behavior of the first part of the processing. It contains one periodically arriving user per house that calls the system for each window, thus arriving every *window shift* time units. The call to the system is parameterized with a distribution of the number of elements contained in the window. The different usage scenarios for the system are generated from the workload. For this purpose, our model generator implements the operation that is first evaluated in the system (windowing the incoming data) and derives the characteristics from the resulting data. The result is a distribution of the number of elements per window for each house. The components additionally contain delays that model the behavior of the system when waiting for new data that advances its timer. For example, Figure 3 shows the measurements and predictions of BL and MSAO for an artificial scenario with 4 houses with 4 plugs. In this case, the earliest time for the first window, spanning elements from 0 to 20 time units, can be processed at 60 time units. This is because the window is created after the first element after the window has been received by the *Median* component, resulting in a delay of $T_1 \geq$ *window size*. The component for creating medians can immediately process the window. The components that group the medians again for the time window have to wait for the first group of the next window to arrive, resulting in an additional delay of $T_2 =$ *window shift*. Additionally, the join has to wait for both sides of the processing to finish and for the next group of elements to arrive, resulting in additional delay of $T_3 =$ *window shift*. The overall collective operation delay is thus at least $T_1 + T_2 + T_3 =$ *window size* $+ 2 \cdot$ *window shift* time units. The result that we get from the simulation of the model is a response time for each of the users providing windows, which is equivalent to the point in time T_W that the whole window has been processed. To calculate the ages of single data elemente, we apply a distribution that we have measured in the data: we generate $[T_W - \text{window size}, T_W]$ measurements that are spaced by the average interarrival time for the house this process represents. For example, for a house for which plugs send data with an average interarrival time of 2 seconds and a

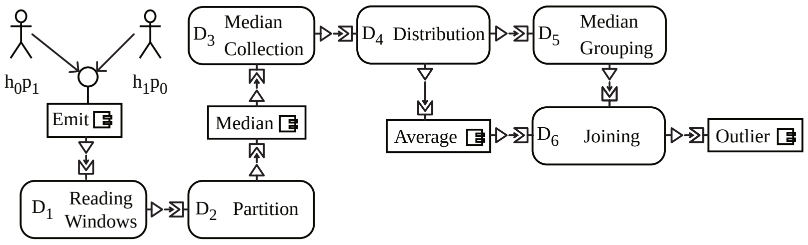

Fig. 4. The model of the system using SAOs. Rectangles are components, rounded rectangles are data channels.

window size of 20 seconds, a window that has finished processing at 60 seconds results in the data age metric $[40, 42, 44, \ldots, 60]$.

Model with Stateful Asynchronous Operations. Figure 4 illustrates the model of the system using our approach. The model separates the operations for collecting and regrouping the arriving data. In summary, the components have the following functions: – *Emit*: emits a sensor reading when called, – D_1: emits group of all plug readings in sliding window, – D_2: partitions readings by plug identifier and emits a collection per plug and sliding window – *Median*: calculates median for each plug – D_3: collects all medians for a time window – D_4: duplicates data item and distributes it to both *Average* and D_5 – D_5: groups the medians per house – *Average*: calculates an overall average across all medians – D_6: joins the overall average and house median groups – *Outlier*: calculates one outlier per average and house median group Each of the parts of the case study system (see Figure 1) is realized by one or more of components in MSAO: system ingress is realized by *Emit*, windowing is realized by D_1 and D_2, *Median* is realized by component Median, *Average* is realized by D_3 and component Average, *Outlier* is realized by D_6 and Outlier.

Comparison of BL and MSAO. Altogether, the BL requires the performance engineer to understand the timing behavior of the system in much detail and to adapt the model in case of changes to the underlying system. If a wall clock with discarding was used, the baseline would have to be modeled differently: we would have to first create a model for the percentage that are discarded by measuring the delay of elements before the windowing step. This percentage would then have to be included in a second model that includes the next components to accurately represent the discarding and the resulting characterization of the windowed group. In case of the MSAO, the mode the component operates in is directly associated with the data channel and can be changed as a model parameter for the element. BL also requires the performance analyst to take the metrics provided by the constructed model, i.e., the processing time of *windows* and derive the actual metric of interest from this metric via an additional step, which further increases the overhead of using BL in comparison with MSAO.

7.5 Calibration

We use the measurements in the training set for the calibration of resource demands. While there are different methods for deriving resource demands for a performance model, we use isolated measurements of the different components in our case study system. We measure both the time and the number of elements for i) calculating the median value per plug, ii) calculating the overall plug median average, iii) calculating the outlier value. We then use linear regression to derive a parameterized resource demand value assuming an empty queue and a processing rate of 1 per time unit. The same calibrated and parametrized resource demands are used for both BL as well as MSAO. This means that the resulting time behavior depends on the data characteristics that arrive at the components, e.g., the correct characterization of the number of smart meter readings in a sliding window, and would be the same for BL and MSAO if both provide the same characterization.

7.6 Results

Example results of the model and MSAO and BL can be seen in Figure 5. In our evaluation, we can see that the models can successfully approximate the measurements of the case study system. The overall accuracy of the models is high due to two factors: i) the active resource delays in the case study system are small to emphasize the collective operation delays, ii) the collective operation delay can replicate the behavior of the system.

We expect both the accuracy of MSAO as well as the accuracy of BL to reduce with more complicated operation pipelines and with a more complex interplay between active resource delays and collective operation delays. However, the evaluation supports our thesis that MSAO can model the collective operation delays that occur in our case study system at the same level as BL. Table 1 and Figure 6 show the results of our evaluation across all 30 scenarios, including the 15 test scenarios. The difference between MSAO and BL is negligible in both M_1 (mean: 0.17 percentage points (pp), median: 0.01 pp) and M_2 (mean: 0.046, median: 0.023). As a result, we see an improvement in the capabilities of the performance modeling approach we presented while retaining the prediction quality.

Table 1. Aggregated metrics M_1 and M_2 for BL and MSAO for all 30 scenarios.

		Relative Error (M_1)			Wasserstein (M_2)		
		Mean	Median	Max	Mean	Median	Max
Training	BL	0.05 %	0.04 %	0.12 %	0.013	0.009	0.037
	MSAO	0.02 %	0.03 %	0.05 %	0.008	0.005	0.029
Test	BL	0.32 %	0.17 %	1.66 %	0.101	0.064	0.311
	MSAO	0.15 %	0.16 %	0.35 %	0.055	0.041	0.142
All	BL	0.19 %	0.07 %	1.66 %	0.057	0.024	0.311
	MSAO	0.09 %	0.04 %	0.35 %	0.031	0.021	0.142

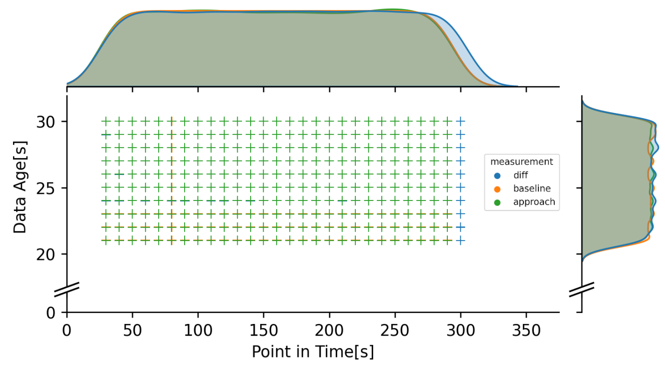

Fig. 5. Exemplary result for DEBS scenario debs-full-4 with window size/shift 10/10.

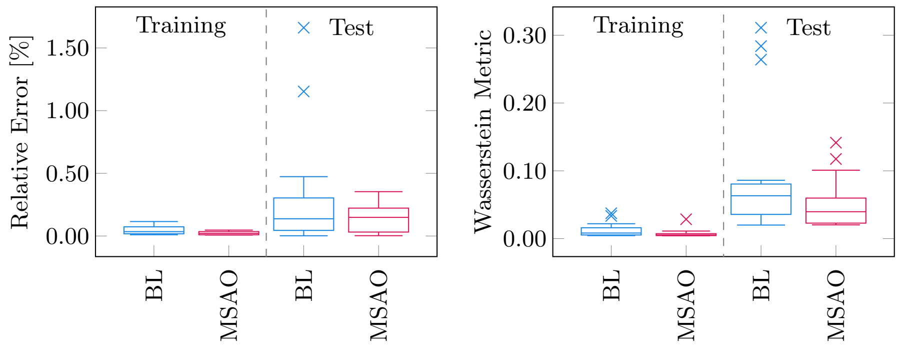

Fig. 6. Aggregated metrics M_1 and M_2 for BL and MSAO for all 30 scenarios.

8 Threats to Validity

In this section, we address threats to validity as proposed by Runeson and Höst [15, sec. 5.2.3] for case-study-based research in software engineering.

Internal validity. Addresses whether all influence factors on the investigated causal relations have been considered. In our case study, we analyzed whether including SAOs in a modeling language allows models that are better aligned with the system architecture without reducing the prediction accuracy (RQ_1). Our focus on collective operation delays helps to minimize other influence factors such as interaction between collective operation delays and active resource delays. Furthermore, we also evaluate and present the behavior for artifical simple scenarios in the training set to show that simple behavior is accurately predicted without unknown influence factors. We also present the factors that influence the alignment of our approach with the system and the baseline with the system in a structured argumentation. One factor that is hard to eliminate is the expertise of the architect modeling the system. We mitigate this factor by creating a baseline that is as sophisticated and accurate as possible to avoid that our approach has an unfair and hidden advantage in the comparison.

External validity. Addresses whether the findings of the case study can be generalized to other cases of interest. According to Runeson and Höst [15, sec. 6.1], case studies does not need to be representative of a population. We aim to increase the external validity by focusing on a case description that comes from the research community. Furthermore, we derive our model elements of interest from operations that are discussed both in the seminal work in the research area ([1]) and are also used in popular frameworks (Apache Beam, Apache Flink).

Construct validity. Addresses whether the metrics that are studied answer the research question. We derive the evaluation goals, questions and metrics according to the established Goal-Question-Metric (GQM) approach [3]. This helps to make the connection between the evaluation and the presented goal transparent. We also use metrics that are established to evaluate performance prediction approaches.

Reliability. Addresses whether the findings depend on the specific researcher that conducted the research. We address this threat by automating the experiment pipeline, from execution and measurement of the case study system to the analysis and derivation of our target metrics and by publishing the whole experiment pipeline. Thus, it is available to other researches who can freely study the experiment setup and also replicate or change the experiments.

9 Conclusion and Future Work

In this paper we have introduced an approach for modeling and simulating stateful asynchronous operations (SAOs) and have demonstrated that we can build performance models that more explicitly represent the behavior of software systems that include SAOs while retaining the accuracy of a state-of-the-art baseline model crafted by an expert performance engineer. The expected benefit of this approach is that software architects are supported in making design decisions for systems that include stateful operations, such as data-streaming systems. One direction in need of further investigation are scenarios where a system becomes overloaded recovers from the load in a phase with reduced load to assess whether models can reproduce the transient behavior of the system. A second direction of research is regarding the modeling language for SAOs. We currently do not restrict their functionality, but there might be cases where it is beneficial to do so to ensure that the simulation results are accurate, for example, if the effects that are modeled rely on events that are very rare and thus are not sufficiently captured by a simulation that only runs for a limited time frame.

Acknowledgements

This work was supported by KASTEL Security Research Labs and by the German Research Foundation (DFG) under project number 432576552, HE8596/1-1 (FluidTrust).

References

1. Akidau, T., et al.: The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc. VLDB Endow. **8**(12) (2015). https://doi.org/10.14778/2824032.2824076
2. Aliabadi, S.K., et al.: Analytical composite performance models for big data applications. J. Network and Computer Applications **142** (2019)
3. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. In: Encyclopedia of Software Engineering - 2 Volume Set. Wiley (1994)
4. Becker, M., et al.: Performance analysis of self-adaptive systems for requirements validation at design-time. In: ACM SIGSOFT QoSA 2013. ACM (2013). https://doi.org/10.1145/2465478.2465489
5. Casale, G., Li, C.: Enhancing big data application design with the DICE framework. In: Advances in Service-Oriented and Cloud Computing – Workshops of ESOCC (2017)
6. Castiglione, A., et al.: Modeling performances of concurrent big data applications. Softw., Pract. Exper. **45**(8) (2015)
7. DICE consortium: Deliverable 3.4 DICE simulation tools (2017), `http://www.dice-h2020.eu/deliverables/`, European Union's Horizon 2020 programme
8. Hummel, O., et al.: A Collection of Software Engineering Challenges for Big Data System Development. In: Euromicro SEAA. IEEE (2018)
9. Jerzak, Z., Ziekow, H.: DEBS 2014 Grand Challenge: Smart homes – DEBS.org, `https://debs.org/grand-challenges/2014/`
10. Jerzak, Z., Ziekow, H.: The DEBS 2014 Grand Challenge. In: DEBS '14. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2611286.2611333
11. Kroß, J., Krcmar, H.: Model-Based Performance Evaluation of Batch and Stream Applications for Big Data. In: MASCOTS. IEEE (2017)
12. Kroß, J., Krcmar, H.: Pertract: Model extraction and specification of big data systems for performance prediction by the example of apache spark and hadoop. Big Data Cogn. Comput. **3**(3) (2019)
13. Maddodi, G., Jansen, S., Overeem, M.: Aggregate architecture simulation in event-sourcing applications using layered queuing networks. In: ICPE'20. ACM (2020)
14. Reussner, R.H., et al.: Modeling and Simulating Software Architectures – The Palladio Approach. MIT Press (2016)
15. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2) (Apr 2009)
16. Sachs, K.: Performance modeling and benchmarking of event-based systems. Ph.D. thesis, Darmstadt University of Technology (2011)
17. Singh, S., et al.: Towards extraction of message-based communication in mixed-technology architectures for performance model. In: ICPE '21. ACM (2021). https://doi.org/10.1145/3447545.3451201
18. Werle, D., Seifermann, S., Koziolek, A.: Data Stream Operations as First-Class Entities in Palladio. In: SSP'19. Softwaretechnik Trends (2019)
19. Werle, D., Seifermann, S., Koziolek, A.: Data stream operations as first-class entities in component-based performance models. In: ECSA 2020. Lecture Notes in Computer Science, Springer (2020). https://doi.org/10.1007/978-3-030-58923-3_10
20. Werle, D., Seifermann, S., Koziolek, A.: Data Set of Publication on Accurate Performance Predictions with Component-based Models of Data Streaming Applications (2022). https://doi.org/10.5281/zenodo.6762128
21. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: SIGMOD. ACM (2006)