Data Stream Operations as First-Class Entities in Palladio

Dominik Werle Stephan Seifermann Anne Koziolek
{firstname.lastname}@kit.edu
Karlsruhe Institute of Technology (KIT)

Abstract

The Palladio Component Model (PCM) is an approach to simulate the performance of software systems using a component-based modeling language. When simulating PCM models, requests only influence each other if they compete for the same resources. However, for some applications, such as data stream processing, it is not realistic for requests to be this independent. For example, it is common to group requests in windows over time or to join data streams. Modeling the resulting behavior and resource demands in the system via stochastic approximations is possible but has drawbacks. It requires additional effort for determining the approximation and it may require spreading information across model elements that should be encapsulated in one place. In this paper, we propose a way of modeling interaction between requests that is similar to query languages for data streams. Thus, we introduce state into models without sacrificing the understandability and composability of the model.

1 Introduction

Performance models are a useful tool predicting a system's response time, throughput or resource utilization in different situations.

The Palladio Component Model The Palladio Component Model (PCM) [5] uses stochastic approximations of the behavior of users and of the system, e.g., for deciding about resource demands and call paths. In the PCM, the load on the system is created by users calling services that the system provides. These service calls then flow through the system and return to the user. The effect that different calls in the system have on each other's timing behavior is well-encapsulated in the competition for resources. The PCM distinguishes two types of resources. *Active resources* are occupied for an amount of time that depends on their processing rate and the resource demand of the call. *Passive resources* are like semaphores which are acquired and released by a call. Apart from resources, the behavior of the user and of components usually does not depend on other *state information* of the system. However, there has been some work on allowing stateful dependencies for Palladio models [3], which increases prediction accuracy.

Data Streaming Applications With the emergence of Big Data applications, two main approaches to data analysis—*batch* and *stream processing*—have been extensively studied and are supported by a variety of frameworks. When both approaches are combined, predicting and balancing the overall performance is particularly interesting.

This paper focuses on extending the PCM in order to be able to build performance models for streaming applications. Streaming applications process a continuous, possibly endless stream of data such as sensor readings. An example for a data streaming application is the continuous monitoring of measurements from an internet of things application. For data streaming applications, it is common to collectively process data that belongs together. This may either mean that the data has similar metadata, e.g., because it is sent by the same sensor, or that the data arrived at similar points in time. Common operations for grouping data are building sliding windows over time and partitioning the data or joining data.

To design such a system, it is, e.g., relevant to find out how the number of devices that provide data to a system and their send rate influence the performance of the system. This question can be combined with insights on how the quality of analyses degrades with increasing aggregation of the data, as, e.g., investigated by Trittenbach et al. [6]. Thus, the analysis of streaming applications can allow early decisions on trade-offs between performance and analysis quality.

2 Modeling Streaming Applications

Intuitively, the PCM may represent data being sent to the system as a call to a system interface. Recurring analyses may either be modeled as separate users that start the analysis in specific intervals or as being triggered when data arrives at the system.

The grouping of data based on its characteristics currently cannot be expressed in the PCM and therefore cannot be derived by the analysis. Thus, it is hard to account for consequential timing dependencies such as a window containing a specific number of elements depending on the incoming data rate. A possible workaround is to stochastically approximate the starting times of analyses and the characteristics of data collections. However, this has the following two drawbacks: a) It requires additional effort for deter-

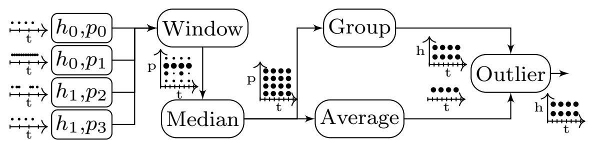

Figure 1: Running example. Small plots show data sent (plug p, household h). Size of circles in windows emitted by Window illustrates number of elements.

mining the approximation, b) It may require spreading information that should be encapsulated in one place (e.g., a view) to multiple model elements. For example, information about requests might be represented in a component behavior description.

Recently, there has been a considerable amount of research on building models of Big Data applications and their performance. There has also been recent work that utilizes the Palladio modeling language to extract performance models for Apache Spark and Hadoop [7]. To our knowledge, current approaches do not explicitly support grouping of data.

3 Running Example

In this section, we introduce a running example to illustrate the benefits of modeling data stream operations as first-class entities. The example is an adaptation of the 2014 grand challenge of the conference on Distributed and Event-Based Systems (DEBS 2014) [4]. The application accepts smart plug readings from different households and calculates an outlier value for each household for shifting time windows.

Structure The example is illustrated in Figure 1. It takes data from N smart plugs which belong to H households. Each plug sends its data at its own rate R_p. They send via a network connection with a latency L_h that may be different for each household. *Window* creates data windows with a given size S for each of the plugs. The windows are created every Δ time units, at $T_i = \Delta \cdot i, i \in \mathbb{N}$, resulting in N windows for each T_i.[1] *Median* creates a median for each window, resulting in N medians for each T_i. *AverageAll* calculates one overall average value of all medians for one T_i. *Group* collects all median values for one household for one T_i. For each of the H households, *Outlier* calculates the ratio of readings of plugs inside the household that are greater than the overall average and emits this value as the outlier value of the household, resulting in H values for each T_i.

Performance We consider the duration from the creation of a date in a sensor to the output of the respective outlier value, *delay*, to be our metric of interest. The utilization of processing, network, and HDD resources might be of further interest. In the following, we only discuss data-streaming-specific influence factors on the system performance.

Modeling Shortcomings While T_i at which windows arrive at *Median* is trivial to determine, the characteristics of the windows—e.g., the number of contained elements—depends on N and R_p. These characteristics could be derived manually and encapsulated in the usage model of an additional periodically arriving process. However, a change in N would require a change in this process. The resource demands of *Median* depend on this contained number of elements. *Group* has to calculate H groups. This has to be encoded in the resource demand or set via a component parameter. This means that the usage model and the resource demand in the repository have to always be consistent. This information is spread across different views, thus breaking the encapsulation of the views. H also influences the number of outliers that have to be calculated. The outlier calculation has to join two incoming data streams, the groups and the overall average. It is currently not possible to model a dependency on data from different calls.

4 Data Stream Operations in Palladio

We propose an inclusion of data stream operations as first-class entities in the PCM, meaning that they may be explicitly expressed as data stream operations and do not require workarounds or approximations. Our approach is centered around *data channels*, which extend event-based communication as introduced into PCM by Rathfelder [2]. Event-based communication allows push-based communication with filtering and distribution of calls and maintains the semantics and analyses of the PCM model. Data channels allow explicit read operations and require an extension of the simulation, thus changing the semantics of the model.

Data Channels A data channel is a queue that can group elements. It specifies a queuing discipline, e.g. *first in, first out* and a capacity. Additionally, items can be configured to be consumed multiple times or only once. Furthermore, the data channel describes, whether a) producers/consumers should block when the channel is full/empty when writing/reading, or b) an item should be discarded, or c) the consumer should return without elements. In the default case, the data channel queues and dequeues single data elements that are written to or taken from the queue.

Emitting and Consuming Data is written to (resp. read from) a data channel via an emit (resp. consume) event action in a service effect specification. These actions also make the characteristics of the retrieved data available in the current execution context, e.g., to specify resource demands that depend on the number of elements in a group of data.

Grouping, Partitioning and Joining We currently support either a) no grouping, b) grouping all

[1]The windows overlap if $\Delta < S$. Then, sensor readings are included in multiple windows.

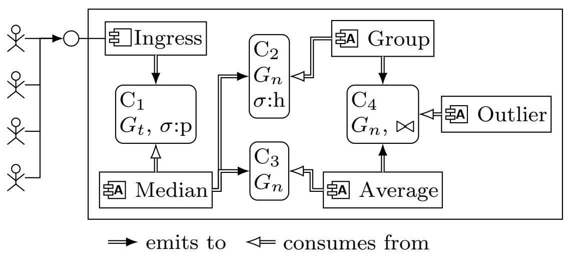

Figure 2: Simplified illustration of the PCM model.

data elements that are currently available in the channel at time of consumption, with an optional minimum or maximum count (shorthand in the following: G_Ω), c) grouping in sliding windows with a given size and shift (G_t), d) grouping according to a key function (G_n). The channel holds back a given number of groups (one by default). If new elements arrive whose keys fit to one of the current groups, they are added to the group. If not, the oldest group is emitted. This can be used to collect groups in an ordered stream of elements, where time is not the emit trigger.

Data channels support partitioning (σ) the data based on a key function that is defined with a stochastic expression. Partitioning is separate from grouping, it is applied after group is created. Here, a suitable key would be the plug id, the household id, or both.

Data channels can join data from different streams if a grouping is defined and two producers write to the data channel ($\bowtie$). Then, a group can only be created, if data from both producers is present in each created group—after partitioning, if applicable.

The operations are similar to the capabilities in the CQL continuous query language [1]. However, they are adapted so they can be described based on metadata instead of concrete values. We currently do not address all types of windowing functions (e.g., session or tumbling windows) but plan to do so in the future and additionally plan to provide a formalization of the semantics of data channels.

The data channel also creates the following characterizations, if applicable: the number of collected elements, the start and end of windows as points of time, the value used in the key function, and statistical values of characterizations of child elements (e.g., household id). We have also created a new characterization, the "birth time" of a date, which may be used to evaluate the *delay*, and for which statistics are also created (e.g., the groups' minimum birth time).

Application to the Running Example Figure 2 illustrates a realization of our running example in PCM. *Ingress* handles the sensor reading ingress and writes data to data channel C_1. There is a usage scenario for each sensor which calls *Ingress* with a characterization of its plug and household id. The windowing of readings is specified in the data channel C_1. All other components are activated by usage scenarios via recurring calls (depicted as). *Median* consumes a window from C_1, possibly blocking until a window is available. It allocates CPU resources depending on the number of elements in the window. It then emits to C_2 and C_3. C_2 groups depending on window start and end and partitions according to the household id. *Group* consumes from C_2 and emits to C_4. C_3 groups by the window start and end. *Average* consumes groups of medians for each time window from C_3. It allocates CPU resources depending on the number of elements in the group and emits to C_4. C_4 joins on the window start/end and allows multiple consumptions of average. *Outlier* consumes from C_4 and specifies an appropriate resource demand.

5 Conclusion

In this paper we presented our approach for modeling data streaming operations in PCM using data channels that can group calls that flow through the system.

We have prototypically implemented our approach into the SimuLizar simulator for the PCM and are currently working on completing the implementation of streaming operations and making it public[2]. Our approach currently modifies the simulation and its semantics. Previous work transformed extended models back to PCM instances [2]. We plan to present such a transformation and a formalization in future work.

In the future, we expect to be able to use our approach together with mappings to concrete technology realizations. We plan to include interfaces for data channels that describe the resource demands of the operations a data channel provides, such as windowing, partitioning and emitting or consuming events.

Acknowledgements This work was partially funded by the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: "Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation".

References

[1] A. Arasu, S. Babu, and J. Widom. "The CQL continuous query language: semantic foundations and query execution". In: *The VLDB Journal* 15.2 (2006), pp. 121–142.

[2] C. Rathfelder. *Modelling Event-Based Interactions in Component-Based Architectures for Quantitative System Evaluation.* The Karlsruhe Series on Software Design and Quality. KIT Scientific Publishing, 2013.

[3] L. Happe, B. Buhnova, and R. Reussner. "Stateful component-based performance models". In: *Software & Systems Modeling* 13.4 (2014), pp. 1319–1343.

[4] Z. Jerzak and H. Ziekow. "The DEBS 2014 Grand Challenge". In: *DEBS '14*. ACM, 2014, pp. 266–269.

[5] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach.* MIT Press, 2016.

[6] H. Trittenbach, J. Bach, and K. Böhm. "On the Tradeoff between Energy Data Aggregation and Clustering Quality". In: *e-Energy '18*. 2018, pp. 399–401.

[7] J. Kroß and H. Krcmar. "PerTract: Model Extraction and Specification of Big Data Systems for Performance Prediction by the Example of Apache Spark and Hadoop". In: *Big Data and Cognitive Computing* 3.3 (2019).

[2] `https://sdqweb.ipd.kit.edu/wiki/PCM_Data_Channels`