Identifying Bottlenecks in a Visualization Platform for Tracing Adaptation Decisions

Martin Pfannemüller Christian Becker
{martin.pfannemueller,christian.becker}@uni-mannheim.de
Universität Mannheim, Germany

Abstract

Measuring the performance of different operations as part of a software system can help to identify performance bottlenecks. This information can be used to improve the performance of the monitored system. In previous work, we proposed a visualization platform for observing and understanding adaptation decisions in self-adaptive systems. In this paper, we apply Kieker as a monitoring solution to measure the runtime of different operations based on the granularity of methods and tasks. As a result, we identified possibilities for improvements in our implementation.

1 Introduction

Today's software systems, such as communication systems, consist of a high number of capabilities and configuration options. With the changing landscape of our networks, communication systems like overlay networks have to adapt themselves to the changing execution contexts. As an example, a wireless sensor network (WSN) can adapt its topology using adaptive topology control based on the position and number of nodes in the system. Concepts from self-adaptive systems research, such as the introduction of an adaptation logic (AL) to a system, can be applied for improving the system's behavior. However, in case a developer achieves to add adaptive behavior to a system, it is not directly observable how and if the system is adapted by the adaptation logic. In previous work, we tackled this challenge of understanding and tracing a self-adaptive system by providing a visualization tool for making adaptation decisions in the domain of communication systems traceable [8, 9]. The tool was evaluated concerning the functional requirements of traceability and extensibility as well as the nonfunctional requirement of responsiveness. The responsiveness has been measured in a high-level aggregating fashion. Hence, this evaluation was not helpful for identifying bottlenecks in the implementation in a fine-grained way. In this paper, we tackle this by applying a well-known performance measurement framework. By that, we measure the performance of different operations of the visualization tool using a prerecorded trace of a WSN with and without taking specific events of the trace into account.

The remainder of this paper is structured as follows. Section 2 presents theoretical background and the system from previous work. Additionally, this section outlines the approach of measuring the performance of our system in a detailed way by applying Kieker, a framework for performance monitoring [6]. The following Section 3 shows the results of our measurements for identifying the performance bottlenecks. Finally, Section 4 concludes this paper.

2 Background and Approach

This section introduces background information about self-adaptive systems and the measured system. Then, the approach for identifying bottlenecks in our implementation is outlined.

Self-Adaptive Systems: Self-adaptive systems adapt themselves to changes in the execution environment and their technical resources [2]. First, they consist of the actual system that should be adapted which is called managed resource (MR). This resource can, e.g., be some software, hardware, or the network. Second, they contain a control loop called adaptation logic (AL). The MR sends sensor information about the execution environment and technical resources to the AL, while the AL uses this information for providing adaptations. These adaptations can be changes of parameters or the architecture of the MR [3].

Dynamic Software Product Lines: One approach for specifying and building self-adaptive systems is to use dynamic software product lines (DSPL) [4]. In this case, the variability of the MR can, e.g., be specified using feature models [1]. A feature model consists of a tree structure representing the variability of a software artifact. This approach has been extended multiple times, including the possibility to model the execution context [5]. By building these context feature models (CFM) representing the reconfiguration possibilities of the software, the possible context states, and constraints, the adaptation behavior can be specified. The system's variability is represented by system features, and the context by context features accordingly. In previous work, we used this specification technique as part of an adaptation logic [7]. A selection of system and context features represents one configuration and state of the system.

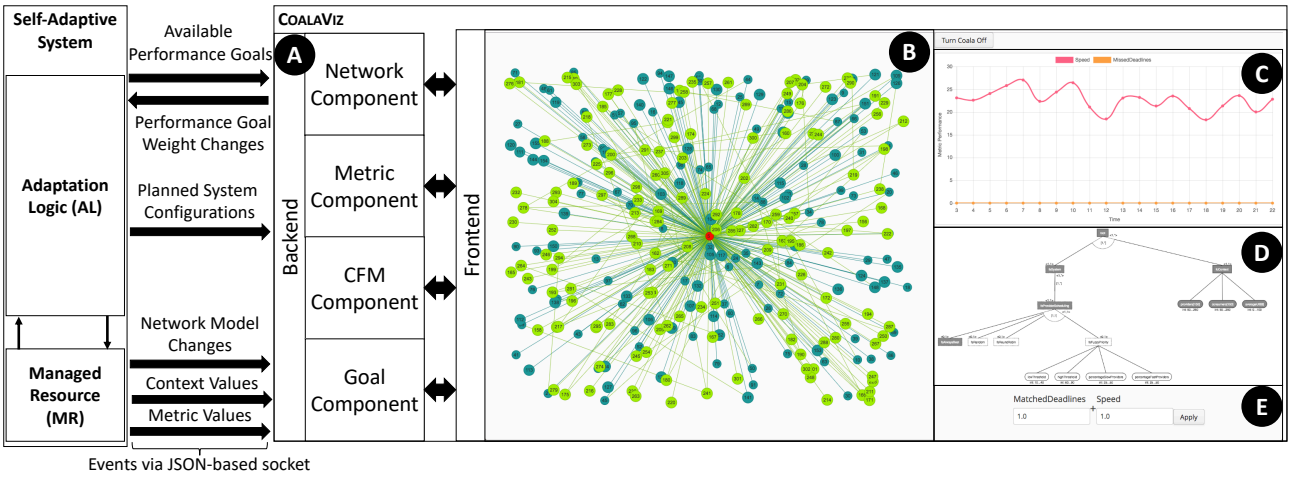

Figure 1: CoalaViz connected with self-adaptive system. Left: Self-adaptive system. Right: CoalaViz with backend A, and visualization consisting of network B, metric C, CFM D, and goal E views [8, 9].

Visualization Tool: After applying the approach from [7], we identified the traceability of reconfiguration decisions as critical for developers and system designers [8, 9]. This led to a visualization tool for self-adaptive communication systems named CoalaViz. Figure 1 shows an overview of the system with the self-adaptive system on the left. The AL and the MR are connected via a socket interface to the backend of the visualization tool A. The views consist of a network view for showing the network topology B, a metric view for plotting nonfunctional metrics C, a CFM view for presenting the context feature model including the selection of system and context features D, and a goal view for setting the weights for the nonfunctional goals E. The system was implemented with Java using Vaadin[1] and Javascript forming a web application. For sending the events to CoalaViz, the system uses a JSON-based socket interface.

Measurement Approach: The quantitative performance evaluation of the visualization tool in [8] revealed low responsiveness in case of many simultaneous events. In this paper, we apply Kieker [6] to analyze the performance of CoalaViz in a fine-granular fashion and to identify the bottlenecks that cause these performance issues. Kieker is an extensible framework for measuring software performance. It is not only capable of conducting performance measurements, but it also can analyze and visualize the monitored data. Additionally, Kieker is able to adapt the monitoring at runtime. It provides multiple ways of monitoring. In our approach, we directly embedded Kieker as a dependency in our code, writing the logs to a temporary directory as described in the user guide[2]. We applied Kieker in methods of the system, beginning from sending events until the changes are actually rendered.

[1] https://vaadin.com/
[2] http://eprints.uni-kiel.de/16537/

3 Results

For our measurements, we employed the same WSN replay used in [8]. The replay consists of different events from a simulated WSN with a running adaptation logic as evaluated in [7]. It simulates 2.5 minutes of the WSN's behavior. Figure 2 shows an overview of the number of events per type. The figure shows that mostly nodes and edges are changing over time. The former change regarding their position, as the nodes are moving, while the latter change as the signal strength updates accordingly. The movement leads to removed edges while some nodes are added, and no nodes get removed. Sixty-nine times a new metric value is sent and four times the adaptation logic sends an updated CFM selection.

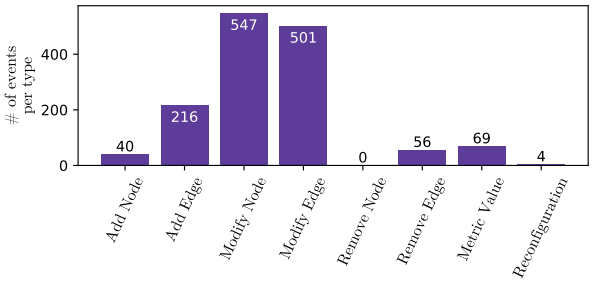

Figure 2: Number of events per event type in the WSN replay. Node and edge events concern B, the metric value event belongs to C, and the reconfiguration event contains a new selection for the CFM D.

As a first result of our measurements, see Figure 3. It consists of averaged measurements without taking the event type into account, including: 1) transmission of the event from the AL to CoalaViz, 2) parsing the JSON structure, and 3) actually handling the event. Handling here means triggering the event-specific frontend code. As the network and the

metric view use Javascript libraries, this triggers the Javascript code that runs asynchronously [8]. Without taking the event type into account, the measurements show, that the transmission time, the JSON parsing, and the handling time per event are not significantly high. For the actual transmission and parsing, there are not many optimization possibilities. However, handling the events is implemented by tool-specific code. Hence, the next step was to measure the handling time of events per event type.

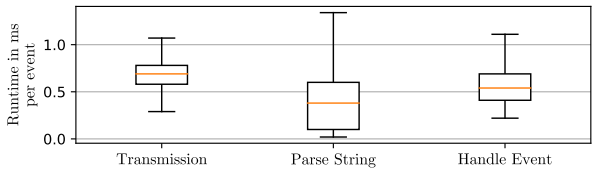

Figure 3: Average runtime of event transmission, parsing, and handling.

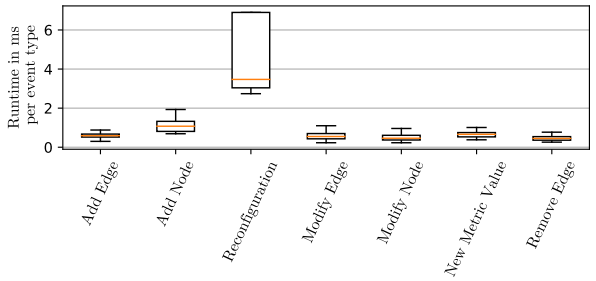

Figure 4: Average handling runtime of events per event type.

The handling times per event type are shown in Figure 4. It confirms the findings in [8], stating the custom-built CFM view is slower than the Javascript views. One reason for this is that it is rendered as an image in the web browser. Thus, even in case of small changes, the whole image has to be rerendered. The measurements indicate that CoalaViz can handle single events in a responsive way. However, the overall measurements in [8] show some spikes in the response time, even at timestamps with a low number of simultaneous events. Based on the fine-grained results of this work, we investigated our code with respect to handling multiple events at the same time. By that, we could identify sequential event-processing, which reduces the overall performance. One way of improvement would be to parallelize the execution. Nevertheless, merely parallelizing this aspect does not work as, e.g., modifying a node depends on the earlier creation of the same node. This could lead to broken events which depend on other events to be rendered first. In future work, we will fix this bottleneck using a queue-based solution.

4 Conclusion

This paper applied the Kieker monitoring framework in a Vaadin-based web application. The measurements show that the render times of the CFM view are higher compared to the other views. With the help of Kieker, we were able to identify sequential processing in our code, which causes bottlenecks and reduced scalability. We were able to apply Kieker using the provided user guide in a basic way. More advanced techniques such as the aspect-based measurement, the analysis, and adaptation possibilities were not tested yet. In the future, we plan to apply Kieker in the self-adaptive system and especially as part of the AL. By that, we are planning to also use Kieker's analysis, visualization, and adaptive monitoring capabilities.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of project A4 of the Collaborative Research Center (CRC) 1053–MAKI.

References

[1] K. C. Kang et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. Carnegie-Mellon University Software Engineering Institute, Nov. 1990.

[2] J. O. Kephart and D. M. Chess. "The Vision of Autonomic Computing". In: *IEEE Computer* 36.1 (2003), pp. 41–50.

[3] P. K. McKinley et al. "Composing Adaptive Software". In: *IEEE Computer* 37.7 (2004), pp. 56–64.

[4] S. O. Hallsteinsen et al. "Dynamic Software Product Lines". In: *IEEE Computer* 41.4 (2008), pp. 93–95.

[5] H. Hartmann and T. Trew. "Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains". In: *SPLC*. IEEE Comp. Society, 2008, pp. 12–21.

[6] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: a framework for application performance monitoring and dynamic software analysis". In: *ICPE*. ACM, 2012, pp. 247–248.

[7] M. Weckesser et al. "Optimal reconfiguration of dynamic software product lines based on performance-influence models". In: *SPLC*. ACM, 2018, pp. 98–109.

[8] M. Pfannemüller et al. "CoalaViz: Supporting Traceability of Adaptation Decisions in Pervasive Communication Systems". In: *PerCom Workshops*. IEEE, 2019, pp. 590–595.

[9] M. Pfannemüller et al. "Demo: Visualizing Adaptation Decisions in Pervasive Communication Systems". In: *PerCom Workshops*. IEEE, 2019, pp. 335–337.