Capacity Planning for Event-based Systems using Automated Performance Predictions

Christoph Rathfelder
FZI Research Center for Information Technology
Karlsruhe, Germany
rathfelder@fzi.de

Samuel Kounev
Karlsruhe Institute of Technology
Karlsruhe, Germany
kounev@kit.edu

David Evans
University of Cambridge
Cambridge, UK
david.evans@cl.cam.ac.uk

***Abstract*—Event-based communication is used in different domains including telecommunications, transportation, and business information systems to build scalable distributed systems. The loose coupling of components in such systems makes it easy to vary the deployment. At the same time, the complexity to estimate the behavior and performance of the whole system is increased, which complicates capacity planning. In this paper, we present an automated performance prediction method supporting capacity planning for event-based systems. The performance prediction is based on an extended version of the Palladio Component Model – a performance meta-model for component-based systems. We apply this method on a real-world case study of a traffic monitoring system. In addition to the application of our performance prediction techniques for capacity planning, we evaluate the prediction results against measurements in the context of the case study. The results demonstrate the practicality and effectiveness of the proposed approach.**

I. Introduction

The event-based communication paradigm is used increasingly often to build loosely-coupled distributed systems in many industry domains including telecommunications, transportation, supply-chain management, and business information systems. Compared to synchronous communication using, for example, remote procedure calls (RPCs), the event-based communication among components promises several benefits, like higher system scalability and flexibility [1]. The deployment of components as well as the connections between producers and consumers of events can be easily changed. However, the event-based programming model is more complex as the application logic is distributed among multiple independent event handlers with decoupled and parallel execution paths. For this reason, predicting the system's behavior and estimating the required infrastructure resources is a complex task. To guarantee adequate performance and availability, systems in today's data centers, are often deployed on server machines with highly over-provisioned capacity [2]. Capacity planning is performed based on data collected on test systems or live systems with a similar setup and workload. Thus, changing the system by adding new components and functionality or changing the workload, often requires expensive and time-consuming load testing. To improve the capacity planning process and thereby the energy and resource efficiency of event-based systems, automated techniques are required that help to estimate the amount of resources required for providing a certain Quality-of-Service (QoS) level. Such techniques help to answer the following questions that arise frequently both at system deployment time and during operation:

- What would be the average utilization of system components and the average event processing time for a given workload and deployment scenario?
- How much would the system performance improve if a given server is upgraded?
- How would a change in the workload affect the system's performance?
- What maximum load level can the system sustain for a given resource allocation?
- What would be the influences of adding new event producers and consumers on the system's performance and resource utilization?
- What would be the performance bottleneck for a given deployment?

Answering such questions requires the ability to predict the system's behavior and performance for a given workload and deployment scenario. Performance prediction techniques for component-based systems, surveyed in [3], support the architect in evaluating different design alternatives. However, they often provide only limited support for modeling event-based communication and automated evaluations of different load scenarios. Both aspects are essential for the capacity planning of event-based systems.

The Palladio Component Model (PCM) [4] is a mature meta-model for component-based software architectures enabling quality predictions (e.g., performance, reliability) at system design-time. As shown in a small proof-of-concept case study [5], PCM can be used to evaluate the performance of event-based systems, however, the modeling effort is rather high, as manually performed workarounds [6] are required to represent event-based communication. In [7], we presented an extension of the PCM which natively supports the modeling and prediction of event-based communication in component-based systems. An automated transformation, which maps the newly introduced model elements to existing PCM model elements, allows us to exploit the available analytical and simulative analysis techniques, e.g., [8], [9], [10], while sig-

This work was partially funded by the European Commission (grant No. FP7-216556) and the German Research Foundation (grant No. KO 3445/6-1)

nificantly reducing the modeling effort by more than 80%. Although the approach was shown to be conceptually sound, so far no validation was presented to evaluate its effectiveness, practicality and accuracy when applied to realistic systems.

In this paper, we present an in-depth evaluation and experience report on the use of our automated performance prediction technique for capacity planning in a realistic context. We present a novel case study of a real-life traffic monitoring system based on the results of the TIME project (Transport Information Monitoring Environment) [11] at the University of Cambridge. We conducted capacity planning for different evolution stages of the system. In various scenarios, we changed the deployment of software components on the hardware infrastructure and/or enhanced the system functionality by adding or substituting components. We evaluate the accuracy of our performance prediction technique using measurements taken on a testbed distributed over up to 10 quad-core machines and consisting of 8 different components each deployed with up to 8 instances in parallel. The prediction error was less than 20% in most cases. As today's systems are often over-provisioned by a factor of 2 or more [2], integrating our approach into the capacity planning process promises significant improvements in terms of resource efficiency. Additionally, the case study shows that our extension and its automation reduces the effort required for performance predictions within the capacity planning process drastically. In summary, the contributions of this paper are: i) An experience report on using an automated PCM-based performance prediction for event-based systems to support the capacity planning process, ii) a novel capacity planning case study of a real-life event-based system considering multiple deployment scenarios and resource allocations, and iii) a detailed evaluation of the accuracy of our approach based on measurements taken in a realistic test environment. To the best of our knowledge, no comparable case studies of similar size, complexity and representativeness exist in the literature. The case study presented here is the first comprehensive validation of our approach in a realistic context.

The remainder of this paper is organized as follows. Sect. II introduces the capacity planning process in general as well as our automated performance prediction approach. Sect. III presents our case study of a traffic monitoring system and the application of the capacity planning process. In Sect. IV, we evaluate the prediction accuracy of our approach and analyze the achieved effort reduction. Next, we present an overview of related work and finally in Sect. VI we conclude with a brief summary and a discussion of future work.

II. Capacity Planning Process

The use of event-based communication in software systems promises better performance and scalability. However, the maximum processable workload is highly dependent on the available infrastructure resources. Due to the decoupling of components in an event-based system, parts of the system can be easily overloaded without any impact on the rest of the system. Nevertheless, in such cases, the system would

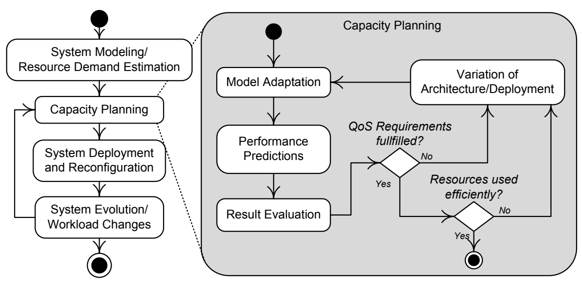

System Modeling/
Resource Demand Estimation
Capacity Planning
System Deployment and Reconfiguration
System Evolution/ Workload Changes
Capacity Planning
Model Adaptation
Variation of Architecture/Deployment
Performance Predictions
QoS Requirements fullfilled?
No
Yes
Resources used efficiently?
No
Result Evaluation
Yes

Fig. 1. Capacity Planning in the Software Life Cycle

be in an unstable state resulting in loss of events or other malfunctions, which are hard to detect. Therefore, determining the resource dimensioning and system deployment to ensure stable processing of the system workload is crucially important and it is the main goal of capacity planning. Additionally, there might be QoS requirements, like maximum event processing time, that must be fulfilled by the system. To guarantee availability, hardware resources are often highly over provisioned [2]. Increasing the efficiency of the system by removing bottlenecks, substituting components, or reducing the required infrastructure while ensuring the availability of the system is an additional aspect of advanced capacity planning processes.

The integration of architecture-level performance prediction techniques into the capacity planning process, as sketched in Figure 1, enables software architects to analyze and evaluate different system configurations and deployments. The specification of the performance model including the estimation of resource demands has to be done only once in the system's life cycle and later adapted to be aligned with the evolution of the implemented system. The resource demands can be estimated by the component developer or, if an implementation of the system is available, based on measurements conducted with this implementation. The performance model itself can be developed manually or generated automatically as done for example in [12]. The performance predictions within the capacity planning process are performed at the model-level, thus the implementation is not affected by the evaluation of different configuration and deployment options. The running system is only reconfigured if the prediction results show that the considered reconfiguration scenario meets the performance and efficiency requirements. As the system evolves (e.g., new components or new hardware is added) or changes in the system's workload are detected, the capacity planning process has to be conducted iteratively.

Each iteration of the capacity planning process consists of several sub-activities. First, if required the system performance model is adapted to reflect the current configuration of the system. Then, by means of the model, a series of performance predictions are conducted in which the load on the system is systematically increased until the maximal sustainable load is reached. The workload variation as well as the execution of the performance predictions is fully automated. We will present more details on our automated prediction approach after introducing PCM in the next section. The results indicate the maximal workload that can be processed by the system. They are additionally used to evaluate if the system fulfills all QoS requirements. In case these requirements are not satisfied with the evaluated system configuration and deployment, the system architect modifies the configuration and/or deployment and repeats the performance prediction process. If the requirements are fulfilled, the system efficiency is evaluated. In case of an efficient resource utilization, the capacity planning iteration ends and the implemented system is reconfigured according to the evaluated model. If efficiency improvements are required, the architect again modifies the system configuration and/or deployment within the model and repeats the performance prediction process. In the following, we first give a general overview of the PCM, then introduce our extension of the PCM that enables the modeling of event-based communication, and finally present the automation of the prediction process.

A. Performance Model

We use the Palladio Component Model (PCM) [4], which is a domain-specific modeling language for modeling component-based software architectures. It supports automatic transformation of architecture-level performance models to predictive performance models including layered queueing networks [9], queueing Petri nets [10] and simulation models [8]. PCM supports the evaluation of different performance metrics, including response time, maximum throughput, and resource utilization. The PCM approach provides an Eclipse-based modeling and prediction tool[1]. Further details and a technical description can be found in [8].

The performance of a component-based software system is influenced by four factors [8]: The implementation of system components, the performance of external components used, the deployment platform (e.g., hardware, middleware, networks), and the system usage profile. In the PCM, each of these factors is modeled in a specific sub-model and thus can be adapted independently. The composition of these models forms a PCM instance.

The *Component Model* specifies system components and their behavior. Components refer to interfaces which are provided or required. The PCM provides a description language, called *ResourceDemandingServiceEffectSpecification* (RDSEFF) to specify the behavior of components and their resource demands.

The *Composition Model* describes the structure of the system. By interconnecting components via their provided and required interfaces, it specifies which other components a component uses.

In the *Deployment Model*, physical resources are allocated to the components of the system. It is required to specify the hardware environment the system is executed on (like processor speed, network links, etc).

The *Usage Model* describes the workload induced by the system's end-users. For example, the workload may specify how many users access the system, the inter-arrival time of requests, and their input parameters.

[1]http://www.palladio-simulator.com

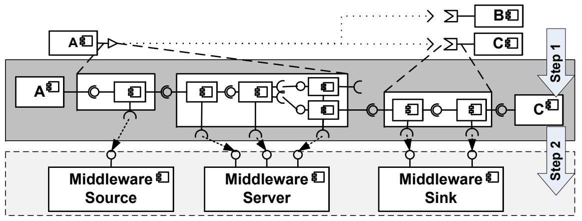


Fig. 2. Automated Model-to-Model Transformation

B. Modeling Event-based Interactions

To enable the semantically correct modeling of event-driven communication, we extended the PCM meta-model with new constructs (e.g., `EventSource`, `EventSink`, `EmitEventAction`) and enhanced the graphical editors to support them. An automated model-to-model transformation transforms these new elements into existing PCM modeling constructs. The transformation allows us to exploit the available analytical and simulative prediction techniques. We substitute the connectors between sources and sinks with several components which cover all aspects of the event processing chain like processing on the client and receiver side, distribution and replication of events, as well as server-side processing. These elements are sketched in Fig. 2. For more details on the component internals we refer to [7]. The transformation includes interfaces to integrate additional components describing the behavior and resource demands of the employed middleware.

C. Automation of the Performance Prediction Process

In the capacity planning process normally different design variations as well as different load situations need to be analyzed and evaluated. In order to reduce the required effort, we have automated the performance prediction process (see Fig. 3). The input of the performance prediction process is a model of the system combined with a specification of the parameter variations. This specification includes the upper and lower bounds as well as the increments of the parameter variations. By means of this specification, the values of model parameters are set. This adapted model is the input of a model-to-model transformation which, as described above and illustrated in Fig. 2 (Step 1), substitutes the new elements of the PCM with elements already supported within the classical PCM. In a second step, components specified in a middleware-specific repository are woven into the PCM model. Depending on the selected prediction technique, this architecture-level model is transformed into a prediction model, e.g., layered queueing network or queueing Petri net, or it is transformed

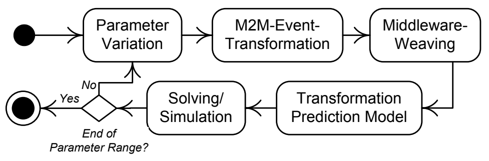


Fig. 3. Automated Performance Prediction Process

into a Java-based simulation code. As a last step, the prediction itself is performed by solving the analytical models or running simulations. If the upper bounds of the considered parameters are reached, the prediction process ends. Otherwise, the process starts again with a new parameter variation. We now present our case study applying the presented capacity planning and performance prediction process in the context of a real-world event-based system.

III. Case Study

The system we study is an event-based traffic monitoring application based on results of the TIME project (Transport Information Monitoring Environment) [11] at the University of Cambridge. It consists of several components emitting and consuming different types of events. The system is based on a novel component-based middleware called SBUS (Stream BUS) [13], which was also developed as part of the TIME project. The SBUS framework encapsulates the communication between components and thus enables easy reconfiguration of component connections and deployment options without affecting the component's implementation. After a short introduction of SBUS , we present the different components the traffic monitoring system consists of. Finally, we conduct the capacity planning process using our PCM-based performance prediction approach in four different evolution scenarios of the system.

A. SBUS Middleware

The SBUS middleware supports peer-to-peer event-based communication including continuous streams of data (e.g., from sensors), asynchronous events, and synchronous remote procedure calls (RPC). In SBUS each component is divided into a wrapper, provided by the SBUS framework, and the business logic that makes up the component's functionality. The wrapper manages all communication between components, including handling the network, registration of event sinks and sources, and marshaling of data.

The basic entity of SBUS is the *component*. Components communicate via *messages* and this is how all data exchange is implemented. Messages are published from and are received by *endpoints*; each endpoint is connected with one or more others. An endpoint specifies the schema of the messages that it will emit and accept. The framework enforces matching of sender and receiver schemas, ensuring that only compatible endpoints are connected. Each endpoint can be a *client*, a *server*, a *source*, or a *sink*. Clients and servers implement RPC functionality, providing synchronous request/reply communication, and are attached in many-to-one relationships. On the other hand, streams of events emitted from source endpoints are received by sink endpoints in a many-to-many fashion.

B. Traffic Monitoring Application

The traffic monitoring application we study consists of 8 different types of SBUS components (see Figure 4).

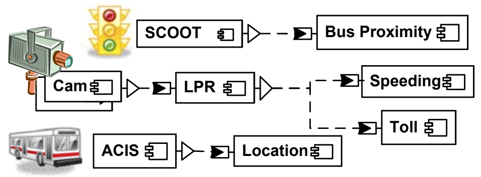


Fig. 4. Overview of Case Study Components

1) Cameras (the "Cam component"): As described in [14], street lamps are equipped with cameras. These cameras only collect anonymized statistical data. In our scenario, cameras take pictures of each vehicle on the street. Each camera is accompanied by an SBUS component, which is responsible to emit the picture combined with position information of the camera and a timestamp as an event to all connected sinks.

2) Licence plate recognition (the "LPR component"): The LPR component can be connected to one or more Cam components. The implementation of our LPR components uses the JavaANPR library [15] to detect license plate numbers of observed vehicles. The recognized number combined with the timestamp and the location information received from the Cam component is then sent out as an event.

3) Speeding Detection (the "Speeding component"): One component consuming the events of detected license plate numbers is the Speeding component. The component calculates the speed of a vehicle based on the distance between two cameras and the elapsed time between the two pictures. [16] reports about the installation of a similar system in London.

4) Toll Collection (the "Toll component"): Another component processing the events emitted by the LPR component is the Toll component. Assuming all arterial roads are equipped with Cam components the toll component determines the toll fee that must be payed for entering the city. The Express Toll Route (http://www.407etr.com) system that is installed near Toronto is an example of such a system which calculates road fees amongst others based on recognized license plate numbers.

5) Bus location provider (the "ACIS component"): The bus location provider uses sensors (in our case, GPS coupled with a proprietary radio network) to note the locations of buses and report them as they change. The component produces a stream of events, each containing a bus ID, a location, and the timestamp of the measurement.

6) Location storage (the "Location component"): The location storage component maintains state that describes, for a set of objects, the most recent location that was reported for each of them. The input is a stream of events consisting of name/location pairs with timestamps, making ACIS a suitable event source.

7) Traffic light status reporter (the "SCOOT component"): In the city of Cambridge, the city's traffic lights are controlled by a SCOOT system [17], designed to schedule green and red lights so as to optimize the use of the road network. The SCOOT component is a wrapper of this system. It supplies a source endpoint emitting a stream of events corresponding to light status changes (red to green and green to red), a second source endpoint emitting a stream of events that reflect SCOOT's measurements of the traffic flow, and two RPC endpoints that allow retrieval of information about a junction.

8) Proximity detector (the "Bus Proximity component"): The Bus Proximity component receives a stream of trigger events reflecting when lights turn from green to red. This stream is emitted by the SCOOT component. Upon such a trigger, the SCOOT component's RPC facility is used to determine the location of the light that just turned red. This is collated with current bus locations (stored in a relational database by the location storage component) to find which buses are nearby.

C. Capacity Planning Study

Thanks to the SBUS middleware, which completely encapsulates the communication between components, the deployment of components as well as the connections between components can be changed with almost no effort. However, as already mentioned in Sect. I, the influence of such changes on the system's performance are hard to estimate. In the following, we apply the capacity planning process introduced in Sect. II to the traffic monitoring system. After introducing the initial performance model, we incrementally extend the model aligned with the system's evolution from a single server deployment to a distributed environment.

1) Performance Model: In this case study, we use the extended version of PCM, which natively supports modeling of event-based communication. The performance model consists of several sub-models described in the following. The complete model is available online[2].

a) Component Models: The parametrization of PCM allows us to specify a repository with reusable components that can be instantiated multiple times if component redundancy is required in the system. To connect the components with the usage model, which specifies the rate of incoming events, we need some additional trigger interfaces. Thus, in addition to the event sinks and sources in Figure 4, the three components ACIS, SCOOT, and Cam provide such additional trigger interfaces. Except for the LPR, the resource demands of the components are nearly constant and independent of the data values included in the event. This allows us to model them as fixed demands in an InternalAction of the respective RDSEFF associated with the component. For each component we measured the internal processing time under low system load and derived the resource demands. Measurements with different pictures showed, that the resource demands required by the LPR component highly depend on the content of the picture. PCM allows to specify parameter dependencies, however, it is not possible to quantify the content of a picture. Thus, we modeled the resource demand using a probability distribution. We systematically analyzed a set of 100 different pictures. For each image, we measured the mean processing time required by the recognition algorithm over 200 detection runs. The standard deviation was less then 2% of the mean value for all measurements. The measurements indicated that the processing of pictures that can be successfully recognized is nearly log-normal distributed ($\mu = 12.23, \sigma = 0.146$). Pictures where no license plate could be detected have a significantly higher but fixed processing time of 109.2ms. To represent this behavior in the RDSEFF of the LPR component, we used a BranchAction. One BranchBehavior contains an InternalAction with the fixed demand for undetected images and the other one contains a log-normal distribution which we fitted to the measurements for successfully detected images.

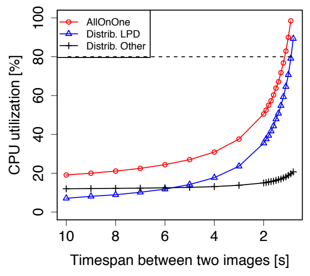
The second component repository we use in our prediction model is the SBUS-specific middleware repository. The modeled middleware components are integrated into the prediction model by the automated model transformation. The repository includes one component representing event sources and one representing event sinks. Both components include a semaphore to model the single threaded behavior of the SBUS implementation. Furthermore, the RDSEFFs include InternalActions to represent the resource demands required within the SBUS-middleware. We instrumented the SBUS implementation to measure the processing time in the different event processing steps.

b) Composition Model: In the composition model, instances of the components in the repository are connected to build the system. The `EventSource` role of a component is connected with the receiving `EventSink` roles. In case of a reconfiguration of the component connections, the model can be adapted by dragging the `Connector` from one component to another. The composition model describes only the connection between components and thus it is independent of the component's deployment on different hardware resources.

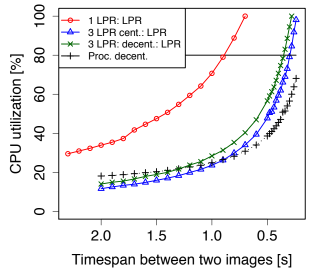
c) Deployment Model: The deployment model describes the allocation of components on individual hardware nodes. It consists of two parts, the resource environment model, which describes the available hardware, and the allocation model, which specifies the mapping of components to hardware nodes. In our case study, the resource model describes our test environment, which consists of 8 `ResourceContainers` each containing one `ProcessingResource` representing the CPU. We selected processor sharing on 4 cores as `SchedulingPolicy`, as all machines in our testbed are equipped with quad-core CPUs. The `ResourceContainers` are connected by a `LinkingResource` with a throughput of 1 GBit/s. The mapping of components to hardware nodes is adapted according to the individual deployment options in the scenarios.

d) Usage Model: The usage model consists of three different types of scenarios, which are executed in parallel. Two `UsageBehaviors` are used to trigger SCOOT and ACIS to emit events. For both behaviors, we specify an `OpenWorkload` with an exponentially distributed inter-arrival time with a mean value of 200ms. Additionally, we introduce a `UsageBehavior` for each street equipped with two cameras. In these behaviors, the two triggering calls of the cameras are connected by a `DelayAction`. With this equally distributed delay, we simulate the driving time of

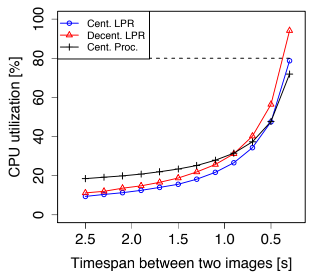
[2]http://sdqweb.ipd.kit.edu/wiki/PCM_Event-Based_Communication



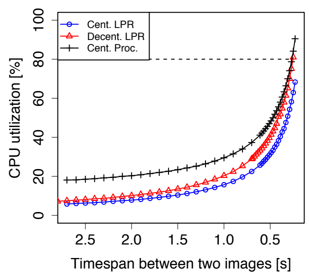
(a) Scenario 2



(b) Scenario 3



(c) Scenario 4 - Old Cam



(d) Scenario 4 - New Cam

Fig. 5. Predicted CPU utilization

a vehicle from the first cam on the street to the second one. Each camera call includes the specification of the image size. Similar to the other behaviors, we use an exponentially distributed inter-arrival time for the first camera. To analyze different load situations, we automatically vary the mean value of this distribution function in a predefined value range.

2) *Capacity Planning:* After introducing the system's components and the performance model, we now apply the PCM-based performance prediction technique to conduct capacity planning. In the real-world, the requirements on the system, the system itself, and the available hardware infrastructure evolve over time. These changes require to evaluate the system and conduct the capacity planning process iteratively. Our case study consists of four different scenarios which cover most of the changes (e.g., change of the system's workload, change of the available hardware resources, modification of a component, or introduction of new components) that require a new iteration of the capacity planning process. We consider four scenarios representing different steps in the evolution of the system.

a) Scenario 1: Base Scenario: The scenario we use as a basis for all other scenarios consists of single instances of SCOOT, ACIS, Location and Bus Proximity. ACIS and SCOOT have a fixed event-rate of 5 events per second. Additionally, one street is equipped with two cameras and two instances of the Cam component which are connected to one LPR component. The detected license plate numbers are processed by the Speeding component. In this scenario, all processing components (i.e. LPR, Speeding, Location, and Bus Proximity) are deployed on one central server. The utilization of this central server needs to be analyzed. The Cam components are running on individual computing nodes which are part of the camera systems mounted on the street lights. In our model, we deployed them on a separate node to avoid any influences on the other components. As ACIS and SCOOT are the connections to other system and thereby to other network segments, they have to be deployed on separate servers for security reasons. In this scenario, there is only one possible deployment option, however, for capacity planning the utilization of this central server as well as the maximal throughput needs to be analyzed subject to the event-rate. The maximal utilization of the CPUs should not exceed 80% to guarantee a stable operation.

In this scenario, we have only one system and allocation model. To analyze and evaluate different load situations, we automatically reduced the timespan between two pictures emitted by the Cam component. The results showed that the system can handle a traffic flow of up to 0.35 seconds between two cars respectively a frequency of $\approx$2.86 cars per second until the limit of 80% resource utilization is reached.

b) Scenario 2: Growing Workload: In this second scenario, two additional streets are equipped with cameras to monitor the traffic, thus the load on the system is increased to a total of six cameras sending images. Additionally, a second server is available. As with the previous scenario, we first analyze the deployment option with all processing components on a single machine, the AllOnOne option, to detect the component which induces most load. We add the new camera components to the performance model and connect them with the LPR component. Again, we specify a automatic variation of the workload induced by the cam components. The bottleneck analysis shows that the recognition algorithm of LPR induces most load on the CPU, so this component is the best candidate to be deployed on the second server. We compare these two deployment options, namely all processing components on one system and LPR separated from the other processing components. In Fig. 5(a) the results of the prediction series are visualized. As the machine hosting the LPR component is still the bottleneck no further optimization is possible in this scenario. Assuming an upper limit of 80% CPU utilization for a stable state, the prediction results show that the AllOnOne deployment can handle up to 0.8 images per second and Cam. The Distributed deployment can handle up to 1 image per second. Thanks to the easy to use graphical editors the required adaptations of the composition and allocation models could be done in less than 10 minutes. After that, the prediction and variation of the event-rate is fully automated.

c) Scenario 3: New Functionality: With the cameras added in the previous scenario, in this scenario all arterial roads in and out of the city centre are equipped with cameras. Based on this data, it is possible to monitor vehicles entering and leaving the inner city. This allows to build up an automated toll collection system, represented by the Toll component. The Toll component is the second component processing the events emitted by the LPR component. It induces additional load on the CPU which was not foreseen in the previous scenarios. To increase the system's throughput, additional hardware is added and it is now possible to run three independent instances of LPR on different nodes. In the first configuration scenario we consider, the new hardware is not used and the LPR component is running separated from all other components similar to the previous scenario. Again, the LPR component is the bottleneck. Based on these results, we evaluated two further deployment options. In both options, three individual instances of LPR are running on different nodes each responsible for the events of two cameras. In the first case, all other components are running on one node (see Figure 6) (centralized deployment) and in the second case Speeding and Toll are deployed with three separate instances and co-located with the LPR instances on the three nodes (decentralized deployment).

The required adaptation effort of the model is slightly higher compared to the previous scenario. However, the adaptation can still be done in less than 20 minutes. The results of the prediction series are visualized in Fig. 5(b). The option with only one instance of the LPR has a maximum throughput of about 1 image per second and camera while the other two options (with three instances) can handle up to 2.5 images per second and camera. Looking at the load balance between the machines hosting the LPR and the machine hosting the other components, the centralized deployment is preferable. The most efficient utilization, i.e., equally balanced CPU utilization, is at an event load with an offset of roughly 0.9 seconds between two images.

d) Scenario 4: Upgraded Hardware: In this last scenario, an additional street is equipped with two cameras. Furthermore, the existing cameras are replaced by a newer and improved model. The new cameras are able to take pictures with higher resolution and improved quality. With the improved quality, the detection error ratio can be reduced from 30% to 5%. It is known, that the resource demands for processing pictures with undetectable license plates is significantly higher than for successfully recognized license plates. However, the resource demands also depend on the image size. In this scenario, the influences of introducing the new camera version on the overall system's performance are evaluated. This evaluation allows to decide if the investment into new cameras will improve the system's performance. Similar to the previous scenario, we evaluate a centralized and a decentralized deployment of the Toll and Speeding components. These two deployment options that are considered, both have four instances of LPR, as a new server node is available.

To represent the new cameras in the prediction model only two model parameters, the size of an image and the probability of an unsuccessful detection, must be changed. Additionally, the new Cam and LPR instances must be added to the composition and allocation models. Nevertheless, the required modeling time is less then 20 minutes. The results are visualized in Fig. 5(c) and (d). In contrast to all other scenarios, the bottleneck in the centralized deployment option with the new cameras is the machine hosting the event processing components and not the machines hosting the LPR components. This means that further replication of the LPR component has no influence on the maximum throughput. Comparing the new and old cameras, the max throughput could be improved slightly by the introduction of the new cameras.

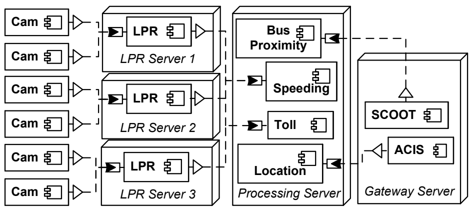


Fig. 6. Scenario 3: Centralized Deployment

IV. Validation

When applying the PCM-based performance prediction technique in the previous section, we assumed that the accuracy of the results is sufficient. In this section, we will validate this assumption and compare measurements in our testbed with the predicted values. Furthermore, we evaluate the effort reduction, that can be achieved by our performance prediction approach.

A. Prediction Accuracy

To evaluate the prediction accuracy, we set up all scenarios described in Sec. III-C in our testbed depicted in Fig. 7. We extended the implementations of SCOOT, ACIS and Cam with configurable and scalable event-generators. The events emitted by SCOOT and ACIS are based on an event stream recorded in the City of Cambridge. The event generator added to the Cam component uses a set of real pictures of different vehicles including their license plates. All event generators have in common that the event-rate can be defined using a configuration file.

A single run of the prediction series simulates about 100,000 pictures and its execution lasts about 3 minutes. On a real system, measuring such a set of data will last up to 5 hours and longer. For this reason, we had to limit the number of experiment runs and workload scenarios. For each scenario, we conducted up to seven experiments which cover the whole range from low to high load on the system. In the following, we present the results of these measurements compared to the predicted values.

1) Scenario 1: Base Scenario: In the base scenario, all event-consuming components are deployed on the same machine. In our testbed, we used three machines. On the first one, we deployed ACIS and SCOOT, on the second one the two Cam components, and on the last one the LPR component together with Speeding, Location, and Bus Proximity. Table I shows the measured and predicted values combined with the

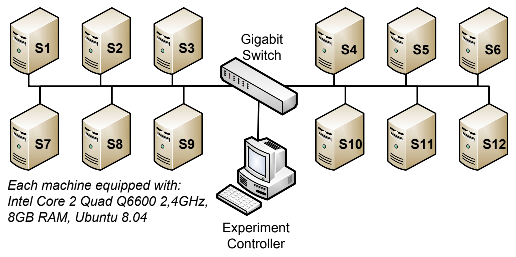

S1
S2
S3
Gigabit Switch
S4
S5
S6
S7
S8
S9
S10
S11
S12
Each machine equipped with: Intel Core 2 Quad Q6600 2,4GHz, 8GB RAM, Ubuntu 8.04
Experiment Controller

Fig. 7. Experiment Testbed

Image rate per Cam [$1/s$]:	0.67	1	1.43	2	3.33
CPU Utilization:					
Measurement [%]:	24.78	33.9	54.64	68.63	92.5
Prediction [%]:	21.8	30.7	42.4	56.9	92.8
Error [%]::	12	9.4	22.4	17.1	0.3
Event Processing Time in LPD:					
Measurement [s]:	0.517	0.52	0.637	0.806	2.409
Prediction [s]:	0.48	0.485	0.503	0.538	2.09
Error [%]::	7.3	6.9	21.1	33.3	13.2

TABLE I
SCENARIO 1: CPU UTILISATION AND EVENT PROCESSING TIME

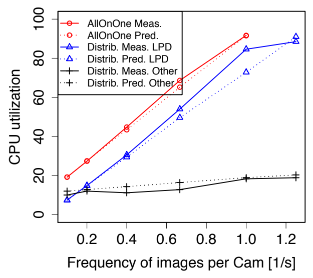
calculated prediction error. Overall, the mean prediction error is less than 20% and the maximal error less than 25% and thus sufficient for capacity planning purposes.

2) Scenario 2: Growing Workload: We set up the AllOnOne as well as Distributed deployment option in our testbed. Figure 8(a) visualizes the predicted and measured mean CPU utilization of the machines hosting the LPR component as well as the machine hosting the remaining components in the distributed deployment. Overall, the mean prediction error of the CPU utilization in this scenario is less than 5%. In both deployment options, the prediction error increases with higher CPU load, which can be explained by caching effects since the algorithm used within the LPR component is very memory-intensive and the high CPU load leads to increasing number of context switches during execution. The measured utilization under the highest load in both options is lower than expected. The analysis of throughput measurements shows that some images were queued up and not processed by the LPR component, if the CPU utilization is higher than 80%. This is an indicator for an overloaded and instable system state. We conducted some more experiments running the system continuously over several hours as well as with an increased event rate. In both cases, the system crashed and completely halted. This confirms our assumption of an overloaded and instable system state.

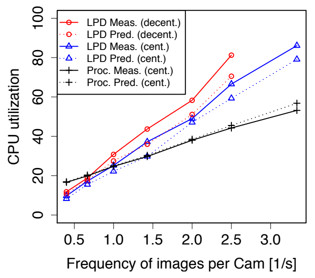
3) Scenario 3: New Functionality: Again, we set up two deployment options in our testbed. In the centralized deployment, the event processing components with exception of the three instances of LPR are deployed on one machine. In the decentralized option, one instance of Toll and one instance of Speeding are deployed with one instance of LPR on the same machine. Figure 8(b) shows the predicted and measured mean utilization of the machines hosting the LPR component for both deployment options. Additionally, it includes the utilization of the machine hosting the processing components in the centralized deployment options. We leave out the values for the decentralized deployment options, as they are independent of the image frequency. Overall, the mean prediction error for the CPU utilization of the machine hosting the LPR component is 11.52% and never exceeded 20%.

Image rate per Cam [$1/s$]:	0.4	0.67	1	1.43	2	2.5	3.33
Measurement (centralized) [s]:	0.47	0.48	0.49	0.55	0.66	0.84	1.99
Prediction (centralized) [s]:	0.41	0.47	0.44	0.43	0.52	0.59	0.96
Error (centralized) [%]:	12.4	2.0	10.0	21.7	21.7	30.4	52.1
Measurement (decentralized) [s]:	0.49	0.48	0.52	0.57	0.68	1.09	-
Prediction (decentralized) [s]:	0.44	0.47	0.44	0.44	0.49	0.73	-
Error (decentralized) [%]:	9.6	2.8	15.0	22.4	27.4	32.2	-

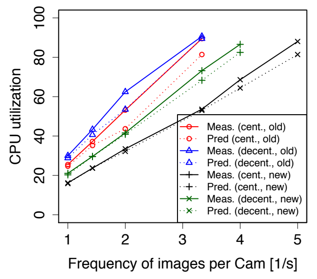
TABLE II
SCENARIO 3: LPR MEAN PROCESSING TIME



(a) Scenario 2



(b) Scenario 3



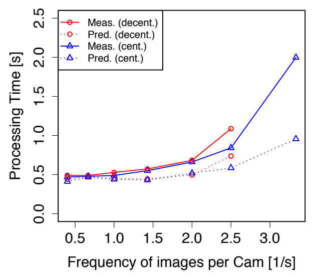
(c) Scenario 4

Fig. 8. Predicted and Measured CPU Utilization

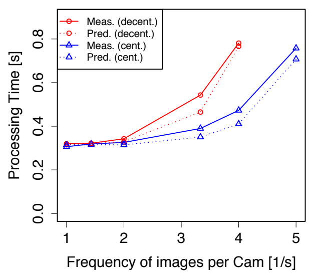
Additionally, we compared the measured and predicted processing time within the LPR component. The results are listed in Table II and visualized in Figure 9(a). Under the highest workload, the decentralized deployment option was overloaded and thus these values are not present in the table and figure. Due to the caching effects, which can not be predicted by the model, the prediction error increases with higher event-rates respectively higher CPU utilization. However, the mean prediction error is still under 20%.

4) Scenario 4: Upgraded Hardware: In this scenario, we set up four different variants of the system, in which we varied between the new and the old version of the cameras by changing the used images and considering again a centralized and decentralized deployment. The results of the measurements and predictions of the mean CPU utilization of the machines hosting an instance of the LPR component are shown in Figure 8(c). Again the prediction error increases with higher load due to the caching effects induced by the memory intensive algorithm of the LPR. However, the mean prediction error is only 5.56%.

We also analyzed the measured and predicted mean processing time within the LPR component. In Figure 9(b), we



(a) Scenario 3



(b) Scenario 4 (New Camera)

Fig. 9. Predicted and Measured Mean Processing Time of LPR

present the processing times of LPR in the scenarios using the improved cameras. The mean prediction error is 5.36% and never exceeded 15%. Similarly to Scenario 2, the measured CPU utilization and processing time in the decentralized deployment option are lower than expected as again events are queued up. The results for an even higher load which completely overloaded the system are not included.

B. Modeling and Prediction Effort

Thanks to the automated prediction process (Sec. II-C) the only manual task that needs to be performed is the adaptation of the system's architecture-level model. In [7], we already demonstrated that the presented PCM extensions combined with the automated model-to-model transformation reduce the modeling effort by up to 80% compared to the use of the original PCM. As already mentioned in the different scenarios, the adaptations of the models could be done with a time effort less than 30 minutes in all cases. The execution of the prediction series is then fully automated. To evaluate the effort reduction achieved with our process automation, we compare the required time to execute the prediction with the time required to conduct equivalent measurements on our test system. One simulation run, which consists of 100000 simulated events, takes about 3 minutes on a MacBook Pro with Core i7 processor and 8 GB RAM. Assuming the highest event-rate of five images per camera per second, this corresponds to a time span of 2.7 hours to collect the same amount of measurements in the testbed. For lower event-rates the required time can be a whole day or more. Thanks to the automated parameter variation, different load situations can be evaluated automatically in less than 1 hour which might require several days of measurements on the test system to obtain the same results.

In summary, the prediction error of CPU utilization and response time is less than 20% in most cases and the maximum error of the always underestimated CPU utilization never exceeded 25%. With this accuracy, the performance prediction can improve the system performance and efficiency significantly given that today's systems are normally over-provisioned by a factor of 2 or more [2]. The presented extension of the PCM combined with the automated model-to-model transformation leads to a significant reduction of the modeling effort by up to 80%. Furthermore, the automated execution of performance prediction series dramatically reduces the required time compared to the execution of measurements on a test system.

V. Related Work

Over the last fifteen years numerous approaches have been proposed for integrating performance prediction techniques into the software engineering process. Efforts were initiated with Smith's seminal work on Software Performance Engineering (SPE) [18]. Since then a number of architecture-level performance meta-models have been developed by the performance engineering community. The most prominent examples are the UML SPT profile [19] and its successor the UML MARTE profile [20], both of which are extensions of UML as the de facto standard modeling language for software architectures. Classical capacity planning techniques based on queueing theory have been studied by Menasce [21]. However, these model do not support an explicit modeling of the software architecture.

In recent years, with the increasing adoption of component-based software engineering, the performance evaluation community has focused on adapting and extending conventional SPE techniques to support component-based systems which are typically used for building modern service-oriented systems. A recent survey of methods for component-based performance-engineering was published in [3].

Several approaches use model transformations to derive performance prediction models (e.g., [22], [23], [24], [8]). Cortellessa et al. surveyed three performance meta-models in [25] leading to a conceptual MDA framework of different model transformations for the prediction of different extra-functional properties [26]. The influence of certain architectural patterns on the system's performance and their integration into prediction models was studied by Petriu [23] and Gomaa [27]. In [23], UML collaborations are used to model the pipe-and-filter and client-server architectural patterns which are later transformed into Layered Queueing Networks.

A method for modeling message oriented middleware systems using performance completions is presented in [28]. Model-to-model transformations are used to integrate low-level details of the middleware system into high-level software architecture models. However, this approach is limited to Point-to-Point connections.

In [29], an approach to predicting the performance of messaging applications based on Java EE is proposed. The prediction is carried out during application design, without access to the application implementation. This is achieved by modeling the interactions among messaging components using queueing network models, calibrating the performance models with architecture attributes, and populating the model parameters using a lightweight application-independent benchmark. However, again the workloads considered do not include multiple message exchanges or interaction mixes.

Several performance modeling techniques specifically targeted at distributed publish/subscribe systems exist in the literature (e.g., [30], [31]). However, these techniques are normally focused on modeling the routing of events through distributed broker topologies from publishers to subscribers as opposed to modeling interactions and message flows between communicating components in event-driven applications.

In [32], a methodology for workload characterization and performance modelling of distributed event-based systems is presented. A workload model of a generic system is developed and analytical analysis techniques are used to characterize the system traffic and to estimate the mean notification delivery latency. For more accurate performance prediction queueing Petri net models are used. While the results are promising, the technique relies on monitoring data obtained from the system during operation which limits its applicability.

VI. Conclusion

In this paper, we presented the application of an automated performance prediction approach in the context of capacity planning for a real-world traffic monitoring system. The performance prediction approach is based on the Palladio Component Model (PCM), which we extended to support the modeling of event-based communication. An automated model-to-model transformation allows to exploit all existing performance prediction techniques supported by the PCM. The presented case study includes different evolution scenarios to which the capacity planning process was applied. A detailed experimental evaluation of the prediction accuracy using a number of different scenarios representing different system configurations and workloads showed the applicability and accuracy of the prediction approach. The prediction error was less than 20% in most cases. Compared to the original PCM approach, our extension reduces the required modeling effort for event-based systems by up to 80%. The automation of the performance prediction process promises a significant time reduction compared to measurements on a test system.

The presented case study covers the most common evolution scenarios of event-based systems in general and not only in the context of traffic monitoring systems. For this reason, the presented work forms the basis to apply the automated performance prediction approach to different event-based systems in industry and research. The approach allows us to evaluate different design options and supports the detection of performance bottlenecks.

The results presented in this paper form the basis for several areas of future work. In the current version of the proposed PCM extension, filtering of events has to be modeled manually in the sinks. We plan to further extend the PCM to specify filtering rules. Furthermore, we plan to work on extracting prediction models automatically at run-time. The resource discovery component (RDC) which is part of the SBUS framework provides methods to determine the connections between endpoints. This information can be used to create the system model. Additionally, we plan to extend the instrumentation we integrated in the SBUS framework making the measured resource demands available during operation. This will allow to extract model parameters dynamically at run-time and will support the use of models for adaptive run-time performance management. Analyzing the influences of caching effects on the system's performance and their consideration within performance predictions is an additional research topic.

References

[1] G. Hohpe and B. Woolf, *Enterprise integration patterns*. Addison-Wesley, 2008.

[2] J. M. Kaplan, W. Forrest, and N. Kindler, "Revolutionizing data center energy efficiency," McKinsey&Company, Tech. Rep., 2008.

[3] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67-8, no. 8, pp. 634–658, 2009, special Issue on Software and Performance.

[4] J. Happe, H. Koziolek, and R. Reussner, "Facilitating performance predictions using software components," *Software, IEEE*, vol. 28, no. 3, pp. 27 –33, may-june 2011.

[5] C. Rathfelder, D. Evans, and S. Kounev, "Predictive Modelling of Peer-to-Peer Event-driven Communication in Component-based Systems," in *Proc. of EPEW'10*, vol. 6342. Springer, 2010, pp. 219–235.

[6] C. Rathfelder and S. Kounev, "Modeling Event-Driven Service-Oriented Systems using the Palladio Component Model," in *Proc. of QUASOSS 2009*. ACM, 2009, pp. 33–38.

[7] B. Klatt, C. Rathfelder, and S. Koounev, "Integration of event-driven communication into the palladio component model," in *Proc. of QoSA 2011*, 2011.

[8] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, pp. 3–22, 2009.

[9] H. Koziolek and R. Reussner, "A Model Transformation from the Palladio Component Model to Layered Queueing Networks," in *Proc. of SIPEW 2008*, 2008.

[10] P. Meier, S. Kounev, and H. Koziolek, "Automated Transformation of Palladio Component Models to Queueing Petri Nets," in *Proc. of MASCOTS 2011*, 2011.

[11] J. Bacon, A. R. Beresford, D. Evans, D. Ingram, N. Trigoni, A. Guitton, and A. Skordylis, "TIME: An open platform for capturing, processing and delivering transport-related data," in *Proc. of CCNC '08*, 2008.

[12] F. Brosig, S. Kounev, and K. Krogmann, "Automated Extraction of Palladio Component Models from Running Enterprise Java Applications," in *Proc. of ROSSA 2009*. ACM, New York, NY, USA, 2009.

[13] D. Ingram, "Reconfigurable middleware for high availability sensor systems." in *Proc. of DEBS 2009*. ACM Press, 2009.

[14] D. Evans, J. Bacon, A. R. Beresford, R. Gibbens, and D. Ingram, "Time for change," in *Intertraffic World, Annual Showcase*, 2010, pp. 52–56.

[15] (2006) Javaanpr - automatic number plate recognition system. [Online]. Available: http://javaanpr.sourceforge.net/

[16] B. Webster, "Average speed cameras mean no escape for drivers," *The Times online*, 2009. [Online]. Available: http://www.timesonline.co.uk/tol/driving/article5645536.ece

[17] P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton, "SCOOT—a traffic responsive method of coordinating signals," Transport and Road Research Laboratory, Tech. Rep. LR1014, 1981.

[18] C. U. Smith, *Performance Engineering of Software Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[19] Object Management Group (OMG), "UML Profile for Schedulability, Performance, and Time (SPT), v1.1," Jan. 2005.

[20] Object Management Group (OMG), "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)," May 2006.

[21] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design*. Prentice Hall, 2004.

[22] M. Marzolla, "Simulation-based performance modeling of UML software architectures," PhD Thesis TD-2004-1, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Mestre, Italy, Feb. 2004.

[23] D. C. Petriu and X. Wang, "From UML description of high-level software architecture to LQN performance models," in *Proc. of AGTIVE'99 Kerkrade*. Springer, 2000.

[24] A. Di Marco and P. Inveradi, "Compositional Generation of Software Architecture Performance QN Models," in *Pro. of WICSA 2004*.

[25] V. Cortellessa, "How far are we from the definition of a common software performance ontology?" in *Proc. of WOSP '05*.

[26] V. Cortellessa, P. Pierini, and D. Rossi, "Integrating Software Models and Platform Models for Performance Analysis," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 385–401, June 2007.

[27] H. Gomaa and D. A. Menascé, "Design and performance modeling of component interconnection patterns for distributed software architectures," in *Proc. of WOSP 2000*, 2000.

[28] J. Happe, S. Becker, C. Rathfelder, H. Friedrich, and R. H. Reussner, "Parametric Performance Completions for Model-Driven Performance Prediction," *Performance Evaluation*, vol. 67, no. 8, pp. 694–716, 2010.

[29] Y. Liu and I. Gorton, "Performance Prediction of J2EE Applications Using Messaging Protocols," *Proc. of CBSE 2005*, 2005.

[30] G. Mühl, A. Schröter, H. Parzyjegla, S. Kounev, and J. Richling, "Stochastic Analysis of Hierarchical Publish/Subscribe Systems," in *Proc of Euro-Par 2009*, 2009.

[31] S. Castelli, P. Costa, and G. P. Picco, "Modeling the communication costs of content-based routing: the case of subscription forwarding," in *Proc. of DEBS '07*, 2007.

[32] S. Kounev, K. Sachs, J. Bacon, and A. Buchmann, "A methodology for performance modeling of distributed event-based systems," in *Proc. of ISORC 2008*, May 2008.