Efficiently Computing Latency Distributions by Combined Performance Evaluation Techniques[*]

Freek van den Berg
University of Twente, CTIT
Enschede, The Netherlands
f.g.b.vandenberg@utwente.nl

Boudewijn R. Haverkort
University of Twente, CTIT
Enschede, The Netherlands
brh@cs.utwente.nl

Jozef Hooman
Raboud University & TNO-ESI
The Netherlands
jozef.hooman@tno.nl

ABSTRACT

Service-oriented systems are designed for interconnecting with other systems. The provided services face timing constraints, the so-called latencies. We present a high-level performance evaluation technique that can be used by a system designer to obtain distributions of these latencies. This technique is capable of capturing nondeterministic, probabilistic and real-time aspects in one go. Under the hood, the technique is equipped with two mechanisms: (i) selection of the right abstraction of the model (to prevent a state space explosion) by evaluating the performance of executing models of different complexities; and (ii) an efficient algorithm in which basic estimates, simulation, and (probabilistic) model checking are combined. We illustrate our approach with an case on image processing of interventional X-ray systems.

CCS concepts

• **General and reference → Performance**; Evaluation;

Keywords

Probablistic model checking; Simulation; Basic estimations; Domain Specific Language; Performance evaluation

1. INTRODUCTION

Service-oriented systems are designed to provide services to other systems in a flexible, dynamic and agile manner [16], such as internet web servers, and image processing systems. Within these systems, services are autonomous, platform-independent entities that perform functions ranging from simple requests to computationally expensive processes.

Besides providing the proper functionality, i.e., returning the right answers to requests, service-oriented systems often need to meet performance constraints, e.g., the system has to reply to a request within a certain time, generally referred to as *latency*. To meet the constraints, service-oriented systems are equipped with multiple resources to process requests.

Techniques to evaluate system performance come in many flavors. *Basic estimations* employ widely used and generic spreadsheets or other high-level models [1] and lead to extremely fast but often inaccurate results, viz., they are not well suited to capture the system dynamics that parallel processing and scheduling bring about [3].

Simulations [14] provide fairly fast results. These explore the system performance via Monte Carlo sampling [15]. Thereby, statistics are used to generalize the observations.

Analytic queuing methods [10] do not only provide both quick results but also accurate results. In return, they often require the distributions to calibrate the model to be memoryless, and can thus only be used for specific systems.

Exhaustive methods, like (probabilistic) model checking [18], do potentially provide the required accuracy and flexibility in model choice. However, they do not scale well and typically require many computational resources and time. They also suffer from the so-called state space explosion.

In previous work, we developed a method to generate latency distributions via iterative probabilistic model checking [18] for iDSL [20, 19], a high-level language and tool chain for performance evaluation of service systems. It delegates low-level performance queries to the Modest toolset [9].

In this paper, we build on this method. First, in the method [18] the user had to determine the right model abstraction manually. We have automated this by evaluating the execution time of iDSL for various model abstractions and then selecting the best model. Second, the method was rather slow because it analyzes the model frequently and exhaustively. We have increased the efficiency by combining the following performance techniques: basic estimations, simulations, and (probabilistic) model checking.

This paper is further organized as follows. Section 2 provides an overview of related work. Section 3 introduces the case study, including the performance model. Section 4 shows two model simplification techniques and Section 5 four performance evaluation techniques. Section 6 contains the performance evaluation tool chain whose results are validated in Section 7. Section 8 concludes the paper.

2. RELATED WORK

Hierarchical Evaluation Tool (HIT, [5]) provides model-based performance evaluation of computing and communication systems. HIT supports several modes of analysis per model type, leading to measures such as average population, throughput and turn around time. Modular Performance Analysis with Real-Time Calculus (MPA, [21]) computes hard lower and upper bounds using event streams.

.

[*]This research was supported by the Dutch national program COMMIT as part of the Allegio project.

VALUETOOLS 2015, December 14-16, Berlin, Germany
Copyright © 2016 ICST
DOI 10.4108/eai.14-12-2015.2262725

Metropolis [2] offers platform-based modeling. It supports model checking and simulation to obtain the worst, best, and average case latency. Both HIT, MPA and Metropolis separate software and hardware (as with the Y-chart philosophy [13]), but do not specifically support latency distributions.

The tagged customer approach (TCA, [8]) numerically computes response time distributions for queuing networks, represented as continuous-time Markov chains (CTMCs). It is a fast and exact measure. Software Performance Evaluation (SPE, [6]) uses a software model (execution graphs) and machine model (queuing networks) for analysis. Both TCA and SPE rely on memoryless models for efficient analysis.

The Palladio framework [4] evaluates performance using Unified Modeling Language (UML) artifacts extended with performance information. Software/Hardware Engineering (SHE, [17]) uses Parallel Object-Oriented Specification Language (POOSL) models. Both the Palladio framework and SHE mainly use simulations for performance evaluation.

We aim for an approach that yields latency distributions, does not generalize observations using statistics, and provides a certain extend of modeling freedom. None of the above approaches satisfies all these requirements.

3. CASE STUDY: BIPLANE IXR SYSTEM

To illustrate our approach, we evaluate the performance of *interventional X-ray* (iXR) systems designed by our industrial partner Philips. iXR systems are used by surgeons while operating a patient. Figure 1 shows an iXR system consisting of a table, arc and display. During surgery, the patient lies on the table with the surgeon standing next to it. X-ray beams are sent between both ends of the arc to record what is happening inside the body of the patient. The result is shown in high quality on the display after a small latency caused by Image Processing (IP). This latency needs to be below a certain threshold to enable hand/eye coordination [11], i.e., the surgeon perceives the images to be in real-time.

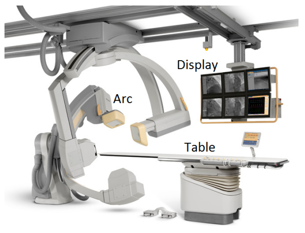


Figure 1: An iXR system

We study the IP latency of so-called Biplane iXR systems with two IP chains that generate 3D images based on two perpendicular planes (named frontal and lateral) of X-ray beams. Traditionally, Biplane systems were implemented with dedicated hardware for each IP chain, but for several good reasons, e.g., physical space, price and energy consumption, we investigate whether Biplane systems with shared hardware are attainable. Below, we model Biplane iXR systems using the iDSL language [20, 19] in six steps.

A ***process*** decomposes service requests into atomic processes. Biplane iXR systems contain two similar processes that each turn X-ray beams into high quality images via a pipeline that decomposes in processes "Noise_reduction" and "Refinement" at its highest level. The former process decomposes into a sequence of five atomic processes, while the latter is a choice between one or two calls of atomic process "Refine", depending on the number of monitors attached to the iXR system. We leave the number of monitors unspecified and model it as a nondeterministic choice. Execution times of each atomic process are estimated by applying the Empirical Distribution Function (EDF, [7]) to a sample of 50 execution times that have been measured on a real iXR system. Consequently, each measurement receives a weight of $\frac{1}{50}$. This yields the following iDSL process:

```
Section Process
  ProcessModel Image_Processing seq {
    seq Noise_reduction {
      atom Pre_processing load EDF from file "pproc"
```

```
      atom Decompose load EDF from file "dcomp"
      atom Spatial_noise_red load EDF from file "snr"
      atom Temporal_noise_red load EDF from file "tnr"
      atom Compose load EDF from file "comp"
    }
    seq Refinement {
      alt { atom Refine load EDF from file "ref"
            seq { atom Refine load EDF from file "ref"
                  atom Refine load EDF from file "ref"
  }}}}
```

Resources are capable of performing one atomic task at a time. In this study, Biplane iXR systems have a single CPU to perform the individual steps of both processes on:

```
Section Resource
    ResourceModel Image_PC decomp { atom CPU rate 1 }
```

Service systems contain services. Biplane iXR systems contain two services, each processing one plane of X-ray beams. Each service decomposes into a process, resource, and a mapping, in accordance with the Y-chart philosophy. In the *mapping*, atomic processes are assigned to resources with a priority scheme. For Biplane systems, we assign all atomic process to resource CPU in a first-in first-out (FIFO) and non-preemptive way, yielding the following system:

```
Section System
  Service Frontal_Image_Processing_Service
    Process Image_Processing
    Resource Image_PC
    Mapping assign { ( * ,CPU) }
      scheduling policy { (CPU,FIFO) }
```

The other service "Lateral_Image_Processing_Service", is specified analogously, using the same process and resource.

Scenarios comprise invoked service requests. Biplane iXR systems process images with inter-arrival times of 40000:

```
Section Scenario
  Scenario BiPlane_Image_Processing_run
    ServiceRequest Frontal_Image_Processing_Service
      at time 0, 40000, ...
    ServiceRequest Lateral_Image_Processing_Service
      at time (0+dspace("offset")),
              (40000+dspace("offset")), ...
```

In order to study the effect of concurrency between the two IP chains, service lateral IP executes after a given offset, depending on the *offset* dimension in the design space. Clearly, offset 0 maximizes concurrency, whereas 20000 minimizes it.

Measures of interest define the metrics to retrieve. We use advanced model checking to return latency distributions:

```
Section Measure
  Measure CDF of ServiceResponse times
    via advanced PTA model checking
```

Finally, the ***study*** comprises a design space with one dimension "offset" that represents four degrees of concurrency:

```
Section Study
  Scenario BiPlane_Image_Processing_run
    DesignSpace ("offset" {"0" "10000" "20000" "30000"} )
```

The aim is to evaluate the performance of the given iDSL model. We use advanced model checking, which is inherently hard (as shown in [18]), mainly because of two reasons. First, the model is complex because it has both nondeterministic and probabilistic traits: nondeterminism oc-

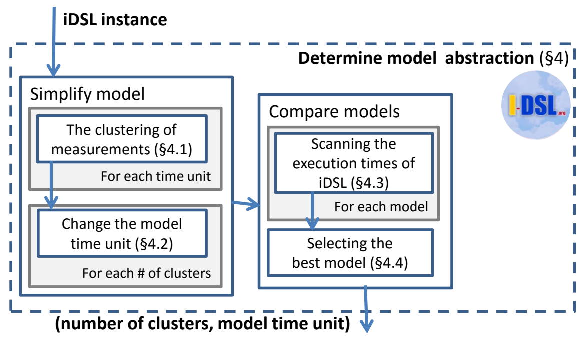

iDSL instance
Determine model abstraction (§4)
Simplify model
The clustering of measurements (§4.1)
For each time unit
Change the model time unit (§4.2)
For each # of clusters
Compare models
Scanning the execution times of iDSL (§4.3)
For each model
Selecting the best model (§4.4)
(number of clusters, model time unit)

Figure 2: iDSL tool chain: determine the model abstraction.

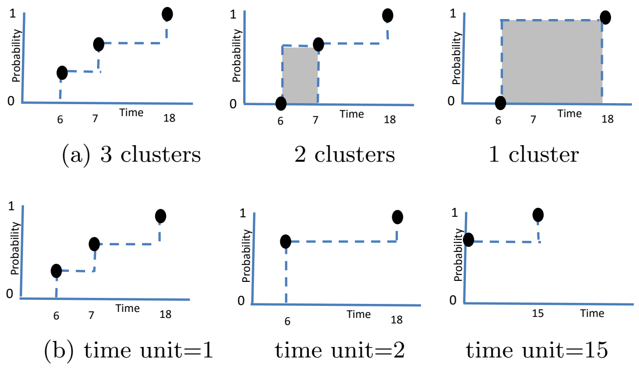


Figure 3: EDF based on measurements $6\mu s$, $7\mu s$ and $18\mu s$.

curs when two atomic processes try to access a resource at the same time, and when to decide whether atomic process "Refine" is executed either once or twice. Probabilism is observed when the execution time for an atomic process is determined as a random selection from the EDF. Second, the model is evaluated frequently and in an expensive way.

4. MODEL SIMPLIFICATIONS

As explained, the model of Biplane iXR systems is complex. We propose an automated model abstraction chain (as depicted in Figure 2) that can be applied to iDSL models, as follows: (i) apply multiple combinations of two model simplification techniques to the iDSL instance, leading to several models; (ii) scan for each of these models the execution time of iDSL for one probabilistic model checking iteration; and (iii) select the model that realizes the best trade-off between model complexity and execution time of analysis.

4.1 The clustering of measurements

The first simplification method is applied to the EDF function of each atomic process, each based on a number of measurements. Measurements are clustered into a given number of clusters, using K-means clustering [12], which has the objective to cluster similar measurements together. For each cluster, its measurements are summarized by an interval of non-deterministic time and all clusters are put together via a probabilistic choice. This reduces complexity by reducing the alternatives for selecting the execution times.

Figure 3a shows a small example based on the three measurement values 6, 7 and 18. Left, it shows the original EDF, which assigns an equal weight of $\frac{1}{3}$ to each of the 3 measurements. When the number of given clusters is greater than or equal to the number of measurements, this original EDF is kept since each measurement is assigned to its individual cluster. In the middle, it shows result of K-means clustering with 2 clusters, viz., measurements 6 and 7 are grouped in one cluster due to their proximity, and 18 in the other. Consequently, 6 and 7 are represented by a non-deterministic time interval, which is graphically depicted as a grey area that covers time range $[6:7]$, and probability range $[0:\frac{2}{3}]$. This grey area represents an ambiguity, namely all distributions that go through this area are possible. Finally, in the case we indicate we only want 1 cluster, the figure on the right shows that all measurements are merged into this single cluster. This leads to a non-deterministic time range $[6:18]$ and probability range $[0:1]$.

The shown ambiguity introduced by the clustering of measurements implies a *loss of information* (loi). This loi per atomic process A, inspired by the objective of K-means clustering [12], can be quantified as follows:

$$loi(A) = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\sum_{x\in A}(x-\mu_i)^2}, \qquad (1)$$

where A is an atomic process represented by a set of measurements, k is the number of clusters, x a measured time, and μ_i the average time of the measurements in cluster i. This measure considers for each measurement the distance to its cluster prototype, viz., the arithmetic mean of the measurements in the cluster, rewarding the clustering of similar measurements. Finally, the loi of the overall process model P is then defined as

$$loi(P) = \frac{1}{|P|}\sum_{A\in P}\frac{1}{|A|}\,loi(A). \qquad (2)$$

4.2 Changing the model time unit

The second simplification method increases the global time unit of the iDSL model. It is again applied to the EDF functions of each atomic process: (i) measurements are divided by the chosen time unit and rounded to the nearest integer value; (ii) performance evaluation is applied; and (iii) the results are multiplied by the chosen time unit. This reduces complexity (less time steps) and precision (rounding errors).

Figure 3b shows an example that is again based on measurements 6, 7 and 18. On the left, the case of time unit=$1\mu s$ is shown, which exactly matches the original EDF, viz., dividing measurements by 1 does not lead to rounding errors. In the middle, the case for time unit=$6\mu s$ is shown. Measurements 6 and 18 are not affected because they are multiples of 6, but measurement 7 induces a rounding error, viz., an integer division of 7 by 6 followed by a multiplication by 6 yields 6 instead of 7. Effectively, measurement 7 is replaced by 6 in the resulting graph, yielding two 6 and one 18 values. Right, we use time unit=$15\mu s$. Measurements 6 and 7 both become 0, whereas measurement 18 transforms into 15. The *loss of precision* (lop) for each measurement x is then:

$$lop(x) = |x - \left\lfloor\frac{x}{t}\right\rceil \cdot t\,|, \qquad (3)$$

where $\lfloor\frac{x}{t}\rceil$ is the nearest integer to $\frac{x}{t}$, t the model time unit. $lop(x)$ ranges from $lop(x)=0$, for $t=1$, to $lop(x)=x$ for $t\to\infty$. The overall lop of a process model P is:

$$lop(P) = \frac{1}{|P|}\sum_{A\in P}\frac{1}{|A|}\sum_{x\in A}lop(x), \qquad (4)$$

Table 1: Execution times (in seconds) of one probabilistic model checking call in iDSL for different model time units and number of clusters, for service Frontal IP, offset=20000.

	$\mathbf{t_{16}}$	$\mathbf{t_{32}}$	$\mathbf{t_{64}}$	$\mathbf{t_{128}}$	$\mathbf{t_{256}}$	$\mathbf{t_{512}}$	$\mathbf{t_{1024}}$	**loi**
$\mathbf{n_1}$	>99	40	7	5	3	3	3	.59
$\mathbf{n_2}$				>99	6	3	5	.44
$\mathbf{n_4}$					**6**	4	6	.33
$\mathbf{n_8}$					10	3	16	.22
$\mathbf{n_{16}}$					7	4	9	.15
$\mathbf{n_{32}}$					7	3	6	.07
$\mathbf{n_{64}}$					8	4	6	0
lop	8.3	17.5	37.1	80.0	180	377	395	

where A is an atomic process model, and P a process model.

Equations (2) and (4) are normalized using $|P|$ and $|A|$ to compare iDSL models with different structures.

4.3 Scanning the execution times of iDSL

In our case study, we combine both model simplifications to define a set of models. Let $\mathcal{M}_{n,t}$ be the simplified model with n cluster segments and time unit t. We then define the following set of 11 × 11 models: $\{M_{n,t} \mid n, t \in \{1, 2, \ldots, 1024\}\}$. Note that the array size and multiplication factors for each dimension are variables in iDSL.

Next, iDSL performs one execution of probabilistic model checking on these models (a "scan"). Table 1 shows the execution times of iDSL in seconds for service Frontal IP and offset=20000. The executions take place starting in the top-right corner at $\mathcal{M}_{1,1024}$ in the Table 1, and going to the left step by step, which is repeated for each line below (i.e., with $n = 2$, $n = 4$, $\ldots$) until $n = 1024$. These execution times can be large, which calls for a stop criterion for each dimension to reduce the overall execution time of this "scan". For the time dimension, we terminate the current execution and skip all remaining executions on the left, when the current execution exceeds a certain time threshold, e.g., 99 seconds. Table 1 shows that models $\mathcal{M}_{1,16}$ and $\mathcal{M}_{2,128}$ exceed this threshold. Hence, models on the left of them have not been evaluated. Moreover, also models positioned on the left-bottom of them have been skipped, since they have more clusters and, thus, are more complex.

For the clustering dimension, the loss of information is used. When it reaches 0, i.e., for $n = 64$ in the case study in which 50 measurements are used, no clustering takes place since each measurement has its own cluster. Hence, increasing the number of clusters, e.g., to $n = 128$ in the case study, leads to a model exactly the same as for $n = 64$.

4.4 Selecting the best model

When the execution times for different models are known as in Table 1, iDSL automatically selects a simplified model as a (user defined) trade-off between five criteria: (i) execution time of one call; (ii) the model time unit; (iii) the number of clusters; (iv) the loss of information; and (v) the loss of precision. To illustrate the effect of combining both model simplifications, we manually (opposed to letting iDSL decide) select model $\mathcal{M}_{256,4}$; It ensures that both model simplifications are applied and it executes fairly quickly.

5. EVALUATION TECHNIQUES

In this section, we present the following four performance evaluation techniques: basic estimates, average behavior, absolute bounds and latency distributions. They will be the components of the tool chain, to be presented in Section 6.

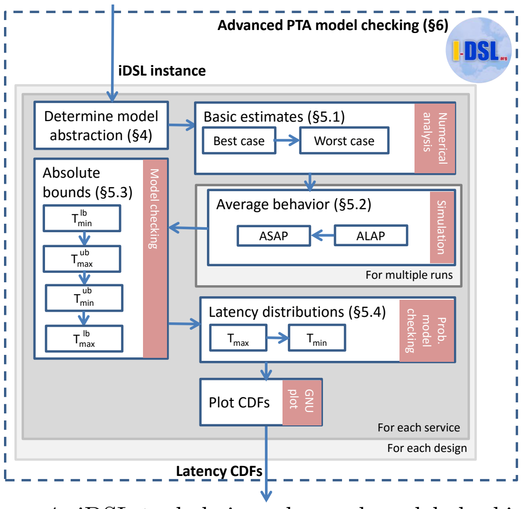


Figure 4: iDSL tool chain: advanced model checking. "Determine model abstraction" is decomposed in Figure 2.

5.1 Basic estimates

Basic estimates are very fast numerical computations that return an optimistic (but maybe inaccurate) bound of either the minimum or maximum latency in a way similar to asymptotic bounds in queueing networks [10]. The result is optimistic because the concurrency between services and processing steps, for resources is not taken into account. Basic estimates directly operate on an iDSL service process, via a recursive algorithm, and return a latency value.

5.2 Average behavior

Average behavior is observed via simulations runs that each return a sequence of latencies. The minimum simulated result is an upper bound for the lower bound, and vice versa. To this end, we apply the MODES simulator [9] to a Modest model, derived from an iDSL model (as in [19, 18]).

5.3 Absolute bounds

Absolute bounds mark the absolute minimum and maximum possible latency. They are a refinement of basic estimates and obtained via model checking on a model in which probabilistic choices are replaced by nondeterministic ones. To this end, we perform a binary search on a given range marked by a minimum and maximum value, in which the MCSTA model checker [9] is iteratively applied to a Modest model derived from an iDSL model (as in [19, 18]).

5.4 Latency distributions

Latency distributions show, for each time, the probability that the latency is less than or equal to a certain value. They are obtained via iterative probabilistic model checking: the MCSTA model checker is applied [9] to a Modest model, automatically derived from the iDSL model, to compute the corresponding probability for each time in a given range.

6. IDSL TOOL CHAIN

In this section, the iDSL tool chain for advanced model checking is introduced (see Figure 4), starting with the model simplification techniques of Section 4, followed by a combination of the evaluation techniques of Section 5, and auto-

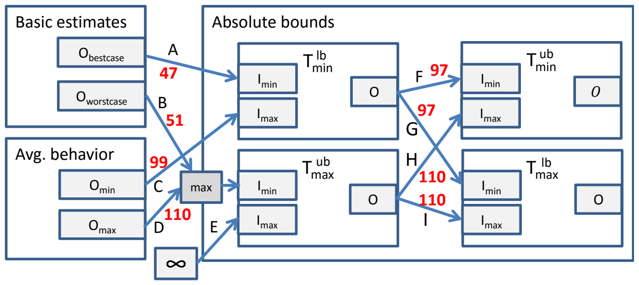


Figure 5: Connections between components "Basic estimates", "Average behavior" and "Absolute bounds". The edges show latencies (red) of service Frontal IP and offset=0.

matically producing latency distributions for each service.

6.1 Tool chain: advanced model checking

Figure 4 shows how the components of the iDSL tool chain connect to return service latencies efficiently, as follows.

First, the model abstraction is determined (of Section 4.4), based on measured iDSL execution times (of Section 4.3).

Second, basic estimates are computed on the basis of iDSL processes (of Section 5.1), yielding a best and worst case.

Third, for average behavior, we perform simulation runs (of Section 5.2). We use 4 runs with as soon as possible (ASAP, [9]) scheduling and as late as possible (ALAP, [9]) scheduling each, of 50 service requests. This is a trade-off between time spent on simulation and model checking later.

Fourth, model checking (of Section 5.3) is performed to compute lower and upper bound latencies via binary searches. Since iDSL models contain nondeterminism whose resolution affects the latency outcomes, we introduce the minimum and maximum time for all resolutions of nondeterminism. Combined, there are four model checking computations, viz., T^{lb}_{min} and T^{ub}_{min} are the lower and upper bound, respectively, for the nondeterminism resolution leading to the minimum latency, whereas T^{lb}_{max} and T^{ub}_{max} refer to to the maximum latency. Section 6.2 explains how basic estimates and simulations are used as input for model checking.

Fifth, latency distributions (of Section 5.4) are computed on ranges determined by the four absolute bounds.

Finally, the latency values are plotted into graphs.

6.2 Estimates, simulations & model checking

Figure 5 shows how the components "Basic estimates", "Average behavior" and "Absolute bounds" of Figure 4 are connected for exchanging latencies. For illustration, we also shows these latencies (in red) for "Frontal IP" (offset=0).

"Absolute bounds" consist of four components (T^{lb}_{min}, T^{ub}_{min}, T^{lb}_{max} and T^{ub}_{max}) that each perform a binary search on an input range $[I_{min} : I_{max}]$ and return a result O. When the input range is wide, many time consuming iterations of model checking are needed to find the result.

Since the best and worst case yield optimistic bounds, Figure 5 shows that the best case (**A**) is a minimum (I_{min}) for T^{lb}_{min}, and the worst case (**B**) a minimum for T^{ub}_{max}.

The minimum "Average behavior" (**C**), obtained via simulations, is a maximum for T^{lb}_{min}, because the lower bound is never larger than any simulated result. Analogously, the maximum "Average behavior" (**D**) is a minimum for T^{ub}_{max}. Since the worst case and maximum simulated result (**B** and **D**) are both a minimum for T^{ub}_{max}, we use the maximum of them for the smallest range size. There is no maximum for the upper bound T^{ub}_{max} (**E**) and it is depicted as infinity here. In the binary search, the infinity is made finite by doubling and evaluating the lower bound repetitively, until a probability of 1 occurs in the evaluation.

Next, T^{ub}_{min} and T^{lb}_{max} have the same ranges, viz., the output of T^{lb}_{min} (**F** and **G**) is a minimum, and T^{ub}_{max} (**H** and **I**) a maximum. These connections are valid, because by definition there is a partial ordering on the output of the "Absolute bounds" components, as follows: $T^{lb}_{min} \leq T^{ub}_{min} \leq T^{ub}_{max}$ and $T^{lb}_{min} \leq T^{lb}_{max} \leq T^{ub}_{max}$ for all services and iDSL models.

In our example we have input ranges $[47 : 99]$ for T^{lb}_{min}, $[110 : \infty]$ for T^{ub}_{max}, and $[97 : 110]$ for T^{ub}_{min} and T^{lb}_{max}, while in previous work [18] we only used the costly range $[0 : \infty]$.

Finally, for the components of "Latency distribution", T_{min} is bounded by the outputs of T^{lb}_{min} and T^{ub}_{min}, and T_{max} by the outputs of T^{lb}_{max} and T^{ub}_{max} (not shown in Figure 5).

Table 2: The execution times of "Frontal IP" on a PC (number of probabilistic model checking calls) of iDSL for components simulation (sim), model simplification (ms), model checking (mc), and probabilistic model checking (pmc)

offset	**sim**	**ms**	**mc**	**pmc**	$\sum$
0	27	499 (16)	758 (27)	305 (7)	1589 (50)
10000	27	646 (26)	333 (18)	135 (7)	1141(51)
20000	26	450 (26)	134 (13)	46 (4)	656 (43)
30000	25	619 (26)	313 (21)	324 (8)	1281 (55)

7. CASE STUDY RESULTS

In this section, we validate the approach by comparing the case study results with corresponding simulation results and assess its efficiency by seeing how quickly it executes.

Approach validity. Figure 6 shows latency distributions of service Frontal IP for four offsets of the case study. The minimum (purple) and maximum (red) latency are generated using the iDSL tool chain of Section 6. The average latency (blue) and 95% confidence interval (black) are based on 5 simulation runs of 200 images. It shows that the minimum and maximum latency encompasses the corresponding confidence interval, for all offsets and probabilities.

Approach efficiency. Previous work [18] has been extended with two model simplification techniques, and three evaluation techniques, viz., basic estimations, simulation, and model checking. They make the performance evaluation approach more efficient, as follows.

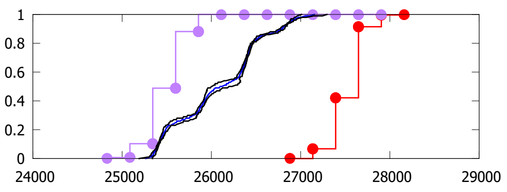
First, Table 2 (*ms*) shows that the automatic simplification techniques of iDSL consume much relative execution time, on a PC (Intel i7-2670QM 2.2GHz, 24Gb RAM). However, without them the system designer has to find the model manually, which is labor intensive and error prone.

Second, basic estimates and simulations make the approach more efficient, viz., executing the iDSL tool chain for service "Frontal_IP" (offset=0), with and without basic estimates and simulations, leads to total execution times of 1937 seconds (61 calls) and 1589 seconds (50 calls), resp.

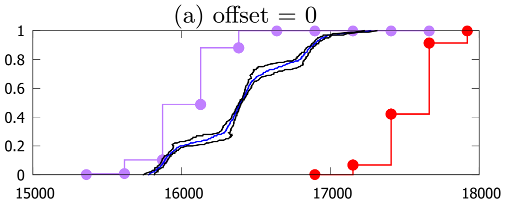
Finally, in Table 2 it shows that model checking calls execute faster on average than probabilistic model checking calls, e.g., for offset=0, (758/27) 28 vs. (305/7) 44 seconds.

8. CONCLUSION

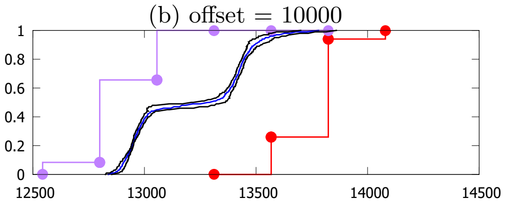
In this paper, we have constructed an automated high-



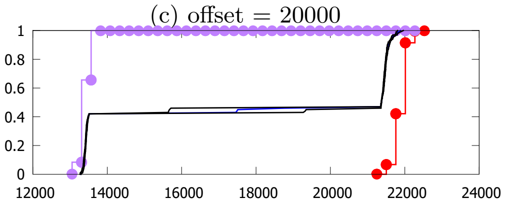
(a) offset = 0



(b) offset = 10000



(c) offset = 20000



(d) offset = 30000.

Figure 6: Latency distributions of Frontal IP for four offsets

level performance evaluation approach and tool chain to obtain latency distributions efficiently, using two mechanisms: (i) model simplifications are selected automatically by comparing the execution times for iDSL of different models; and (ii) different performance evaluation techniques, viz., basic estimations, simulation, and (probabilistic) model checking, constitute an efficient algorithm when combined.

9. REFERENCES

[1] R. Ammar, M. Farid, and K. Yetongnon. A spreadsheet performance approach to integrate a modeling hierarchy of software systems. In *SMC*, pages 847–852. IEEE, 1989.

[2] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003.

[3] T. Basten et al. Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems. In *Model-Based Design of Adaptive Embedded Systems*, pages 189–244. Springer, 2013.

[4] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[5] H. Beilner, J. Mater, and N. Weissenberg. Towards a Performance Modelling Environment: News on HIT. In *Modeling Techniques and Tools for Computer Performance Evaluation*, pages 57–75. Plenum Press, 1989.

[6] A. Bertolino and R. Mirandola. Software performance engineering of component-based systems. In *WOSP*, pages 238–242. ACM, 2004.

[7] C. Forbes, M. Evans, N. Hastings, and B. Peacock. *Emperical Distribution Function*, pages 79–83. John Wiley & Sons, Inc., 2010.

[8] M. Grottke, V. Apte, K. Trivedi, and S. Woolet. Response time distributions in networks of queues. In *Queueing Networks*, pages 587–641. Springer, 2011.

[9] A. Hartmanns and H. Hermanns. The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In *TACAS*, volume 8413 of *LNCS*, pages 593–598. Springer, 2014.

[10] B. Haverkort. *Performance of computer communication systems - a model-based approach.* Wiley, 1998.

[11] J. Johnson. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules.* Elsevier, 2010.

[12] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient K-means clustering algorithm: analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, 2002.

[13] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In *ASAP*, pages 338–349. IEEE Computer Society, 1997.

[14] A. Law and D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, 1982.

[15] N. Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[16] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[17] B. Theelen, O. Florescu, M. Geilen, J. Huang, P. van der Putten, and J. Voeten. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *MEMOCODE*, pages 139–148. IEEE Computer Society, 2007.

[18] F. van den Berg, J. Hooman, A. Hartmanns, B. Haverkort, and A. Remke. Computing Response Time Distributions Using Iterative Probabilistic Model Checking. In *Computer Performance Engineering*, volume 9272 of *LNCS*, pages 208–224. Springer, 2015.

[19] F. van den Berg, A. Remke, and B. Haverkort. A Domain Specific Language for Performance Evaluation of Medical Imaging Systems. In *MCPS*, volume 36 of *OASICS*, pages 80–93. Schloss Dagstuhl, 2014.

[20] F. van den Berg, A. Remke, and B. Haverkort. iDSL: Automated Performance Prediction and Analysis of Medical Imaging Systems. In *EPEW*, volume 9272 of *LCNS*, pages 227–242. Springer, 2015.

[21] E. Wandeler. *Modular performance analysis and interface based design for embedded real time systems.* PhD thesis, ETH Zurich, 2006.