Catching the Response Time Tail in the Cloud

Sebastiano Spicuglia*, Mathias Björkqvist†, Lydia Y. Chen†, Walter Binder*
*Università della Svizzera italiana (USI) – Faculty of Informatics
†IBM Research Zurich – Cloud & Computing Infrastructure

***Abstract*—As modern service systems are pressured to provide competitive prices via cost-effective capacity planning, especially in the paradigm of cloud computing, service level agreements (SLAs) end up becoming ever more sophisticated, i.e., fulfilling targets of different percentiles of response times. However, it is no mean feat to predict even the average response times of real systems, or even abstracted queueing systems that typically simplify system details, and it gets even more complicated when trying to manage SLAs defined by various percentiles of response times. To efficiently capture these different percentiles, we first develop a novel and autonomic methodology – termed *Burst Based Simulation*, which combines burst profiling on real systems with complex, state-dependent simulations. Moreover, based on our methodology, we construct an analysis on SLA management: the prediction of SLA violations given a certain request pattern. We evaluate our approach on two types of service systems, virtualized and bare-metal, with wide ranges of SLAs and traffic loads. Our evaluation results show that our methodology is able to achieve an average error below 15% when predicting different response time percentiles, and accurately capture SLA violations.**

I. Introduction

While computing clouds are becoming the standard platforms for service systems due to their advantages of ease-of-management, service providers also encounter new challenges such as fast workload dynamics and the complex cost structures of clouds [1, 2]. To provide competitive services in terms of price-quality ratios, providers increasingly develop sophisticated service level agreements, SLAs, for users, i.e., guaranteeing certain percentiles of service response times by target values [3]. Typically, violating SLAs can result in immediate financial loss, as well as long term user churn [4]. As a result, it is very important for providers to dimension their resources in the most cost effective way without causing SLA violations. The very first step in managing such sophisticated SLAs is to capture the response times of different percentiles accurately and efficiently.

Predicting response times has long been a research challenge for real systems [5, 6, 7] and synthetic queueing systems [8, 9, 10, 11] . On the one hand, the former addresses the overall system complexity, e.g., parallelism of applications and various hardware specifications, by collecting a large number of low level statistics [12, 13], with a focus solely on the average response time. Overall, the service response times depend on the concurrency provided jointly by the parallelism of system components and applications. On the other hand, after greatly simplifying the system architecture and application behavior, the latter can rarely tackle such difficult problems exactly, especially for bursty requests. Approximation techniques are applied to derive the distribution of entire response times, including all ranges of percentiles. Indeed, bursty requests cause not only intractability, but also degrade the tail response times tremendously for cloud systems [14]. Essentially, it is extremely challenging to predict the response times of different percentiles specified in SLAs, especially for real systems, and unfortunately existing methodologies only provide partial solutions.

In this study, we address the question of how service providers can efficiently predict SLAs based on response time percentiles.Our objective is to develop a plug-and-play methodology, which is accurate in predicting SLAs and is easy to use for service providers with minimum system overhead. We propose a two-phase approach combining profiling on real systems with simulation-based predictive models. For predicting SLA violations, we consider extensive combinations of SLA definitions, i.e., w^* is the target value of the α^{th} percentile of response times. To evaluate the efficiency of the proposed methodology, we use two types of service systems, namely spell check and wikipedia, deployed on different architectures.

In particular, to capture the impact of concurrent and bursty requests on response times, we first propose a novel concept termed a *Burst Profiling Set*, which profiles systems by sending concurrent bursts of requests in batches of different sizes. Through the use of a burst profiling set, we are able to understand how systems concurrently execute bursts that are typically in the order of a few hundred ms in duration, and further derive system execution rates that are shown to be state-dependent. The second step is to develop a state-dependent simulation model, which adopts the execution rates derived from a burst profiling set to process requests taken from synthetic or real traces. The response time percentiles are then predicted via multiple simulation runs. Our exhaustive evaluation results, both on virtualized and bare-metal systems, show a high accuracy in predicting a wide range of response time percentiles, SLA violation, and maximum sustainable throughput.

Our contributions are twofold: (1) we provide a novel and easy-to-use methodology, which is able to predict response times of different percentiles of real service systems. (2) We derive a critical analysis for managing sophisticated SLAs, i.e., SLA violations given w^* as the target value of the α^{th} percentile of response times.

The rest of this paper is structured as follows: Through experimental results, Section II shows the complex relationships between arrival rates and response time percentiles. In Section III we develop a methodology for estimating the entire distribution of the response times of a system. Our analysis on SLA violations is presented in Section IV. In Section V we evaluate the proposed methodology with a set of extensive experiments. Section VI presents the related work, and Section VII concludes the paper.

II. Motivation

In this section we illustrate the challenges of understanding the complex relationship that exists between arrival rates and the percentiles of the response times by running a set of experiments. We consider two different systems: *sys1*, a remote spell-checking service based on an event-driven architecture, and *sys2*, a multi-threaded web server running a clone of Wikipedia (see Section V-A for further details of the setup). We load both systems with request arrival rates of increasing intensity. For each load condition, we measure the 75^{th}, 80^{th}, 90^{th} and 95^{th} percentiles of the response time, and plot the results in Figures 1a and 1b. From the data, we see that the relationship between the request arrival rates and response time percentiles is highly system dependent and non-linear. For sys1 there is a 2.5-fold increase in the percentile at high rates with respect to low rates, while for sys2 it is possible to see a 16-fold increase. This effect can be due to a multitude of causes, e.g., different software architectures for the two systems, or the resources required for processing a request. In cloud environments, the predictability can be further obscured by e.g., virtualization or heterogeneity of the underlying hardware. As confirmed by our experiments, the event-driven architecture of sys1 provides more robust performance than the multi-threaded architecture of sys2 at high load [15]. Moreover, Figure 1a shows how difficult the estimation of the percentiles is. In the range $[150, 180]\, req/s$ there is a highly non-linear behavior that is hard both to predict and explain.

III. Methodology

In this section we first introduce the concept of *Burst Profiling Set*: a data set, containing the outcome of request bursts of different sizes, that captures the complex nature of a real system. Then, we develop a predictive model that takes as input a workload (defined as inter-arrival times between requests) and outputs the estimated empirical distribution of the response times. Finally, we outline the structure of a state-dependent simulator that can be used to run the proposed model. Figure 2 outlines the architecture used to implement our methodology.

The rest of the section is structured as follows: Subsections III-A and III-B describe the two phases; and Subsection III-C highlights the limitations of our approach.

A. Burst Performance Set

We use bursts of requests (i.e., requests hitting the system all at the same time) to characterize the system and consolidate all its properties (e.g., service demand, scheduling policy, request parallelism level, number of cores, thrashing effects) in a format easy to store and process. For each burst of size n, we save the completion time of all its n requests as a tuple:

$$burst_n = < t_1, t_2, ..., t_n >$$

where t_i is the completion time of the i^{th} request. To stress the system with different load conditions (i.e, different number of simultaneous requests), we send bursts of different sizes. We use a threshold on the completion time to find the maximum size N (e.g., a completion time 5x higher than the service demand). Moreover, to accommodate fluctuations in the behavior of the system, we send each burst size r times. We refer to the i^{th} repetition of a burst of size n as $burst_{n,i}$. We refer to the collection of all the bursts as the *Burst Performance Set*:

$$< burst_{1,1}, ..., burst_{1,r}, ...burst_{N,1}, ..., burst_{N,r} >$$

The information contained in a burst performance set reveals many of the properties of a system. Figures 3a and 3b show the burst performance set for two different systems, providing an example of the information contained in such sets (see section V-A for the details of the systems). A burst of size 1 can be seen as the service demand of the system: $25ms$ for sys1 and $270ms$ for sys2. Analyzing the relationship between completion time and burst size, it is possible to estimate the parallelism level of the system: for sys1 there is a significantly increase in the execution time after $size = 6$, whereas for sys2 this already occurs after $size = 4$. Moreover, comparing $size = 7$ and $size = 8$ for sys1, it is possible to glimpse a limited processor-sharing level of 4. The higher burst sizes quantify both the queuing time of the requests and the thrashing effects due to the contention. Another interesting property of a burst performance set is that it can be collected very quickly. The time to gather a burst depends both on the size and the completion time of the requests: The average time to send a burst of 25 requests is $134ms$ for sys1 and $933ms$ for sys2.

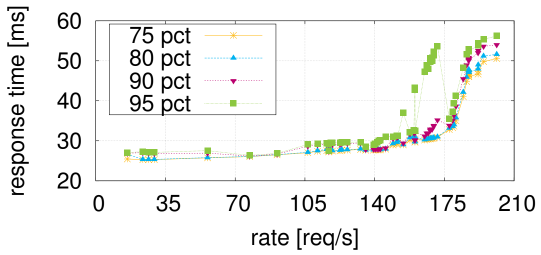
B. Predictive model

To estimate the distribution of the response times, we develop a predictive model that takes as input a stream of inter-arrival rates and a burst performance set, and outputs a stream of response times. Internally, the state of the model is defined by the set of requests currently being executed. In the model, requests are processed with a rate that depends on the number of concurrent requests. In more detail, the state of each request is defined by following tuple: $< start, demand, executed, rate >$, where:

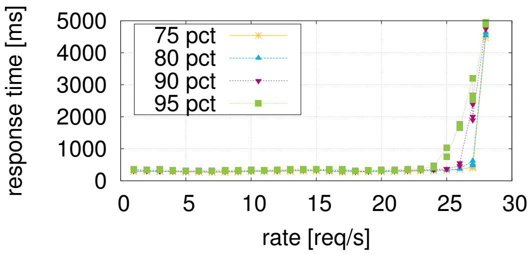
- $start$ is the time when the request joins the system.
- $demand$ represents the number of seconds needed by that request to complete its execution. When a request arrives, this field is set to the value of a randomly chosen burst of size 1.
- $executed$ keeps track of the time spent executing the request. When $executed = demand$, the request leaves the system and the model outputs a response time sample equal to $now - start$, where now is the current simulator time.
- $rate$ is a number in the range $[0, 1]$ and represents the slowdown experienced by the request, e.g., overhead due to thrashing. If during an interval of length Δ the rate has a constant value r, the $executed$ field is updated with the following statement:

$$executed = executed + \Delta * rate \qquad (1)$$

The novel part of the model is how the rates are computed from a burst performance set, shown in Algorithm 1. To explain the algorithm, we use an example of a burst of size 7 from Figure 3a. As a first step the simulator randomly draws one burst of size $n = 7$ out of r samples (Line 2), with the values $< 30ms, 42ms, 30ms, 42ms, 30ms, 30ms, 30ms >$ in our example. In this burst, the first request leaves the system



(a) sys1



(b) sys2

Fig. 1: Relationship between load intensity and percentiles of the response time for two systems.

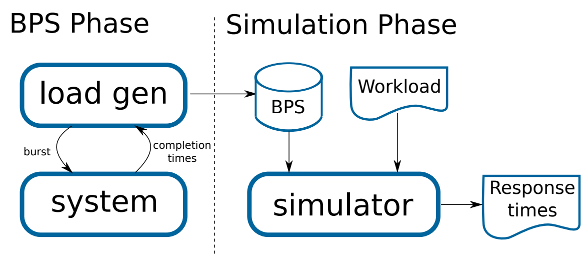


Fig. 2: Architecture employed to estimate response times

after $30ms$, meaning that the state "7 requests are running simultaneously" has a duration of $30ms$ (Lines 3 and 4). The rates can be computed considering how many seconds of execution the system processed for each request during such a state. From a burst of size one (Line 5), we know that the service demand of the system is $25ms$. Therefore the first 4 requests have a rate of $^{25}/_{30}$. The situation is different for the last two requests: They complete in $42ms$, meaning that after the first $30ms$, they run for a further $12ms$ (variable $extra$ at Line 7). Since the service demand of this system is $25ms$, we can conclude that during the first $30ms$, the system processed at least $13ms$ for these two requests. Therefore their rate is $^{13}/_{30}$ (Line 8). In the algorithm all the requests are analyzed in the same way by the for loop. In our example, the first 5 requests are just a special case where $extra = 0$.

Algorithm 1 Rates of n concurrent requests.

```
 1: function updateRates($n$)
 2:     $burst \leftarrow draw\_burst(n)$
 3:     $sort(burst)$
 4:     $duration \leftarrow burst_1$
 5:     $demand \leftarrow draw\_burst(1)$
 6:     for $i$ in $1..n$ do
 7:         $extra \leftarrow burst_i - duration$
 8:         $request_i.rate \leftarrow \frac{demand-extra}{duration}$
 9:         if $request_i.rate < 0$ then
10:             $request_i.rate \leftarrow 0$
11:         else if $request_i.rate > 1$ then
12:             $request_i.rate \leftarrow 1$
13:         end if
14:     end for
15: end function
```

Since the model described so far is easy to simulate, we rely on a simulation-based approach to estimate the distribution of the response times. In each state, the simulator computes two numbers Δ_{forw} and Δ_{back}. The former determines the time of the next arriving request and is drawn from the input stream. The latter is the next completion time assuming that no new request joins the system in the meantime. Δ_{back} is computed as $min_{r \in requests}(\frac{r.demand - r.executed}{r.rate})$. The simulator updates the state of the model based on which event (arrival or completion) happens next. Internally the $executed$ and $rate$ fields of each request are updated as described by Equation 1 and Algorithm 1, respectively. In the following of the paper we refer to the approach described above as *Burst Based Simulation*. We implemented such a simulator as a Java application in less than one thousand lines of code.

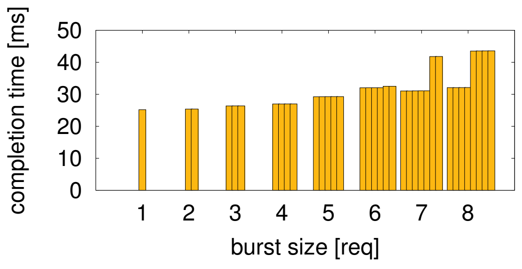
C. Limitations

The approach we propose has two main limitations. The simulator assumes that all the requests are equal, as the rate depends solely on the number of requests and not also on their mix [16, 17]. This could be a limitation for workloads with different classes of requests. Currently we are limited to handling a multi-class workload as a uniform workload with very fluctuating requests. The other limitation is that once we gather a burst performance set we can simulate just a certain kind of stream of arrivals. In the case of very high frequency arrivals, the simulator may reach a number of the concurrent requests higher than the maximum burst size contained in the burst performance set. In such a scenario, the algorithm shown in Algorithm 1 would fail at Line 1. A solution could be estimating the missing burst sizes based on the value already stored in the burst performance set, but we leave this exploration for future work.

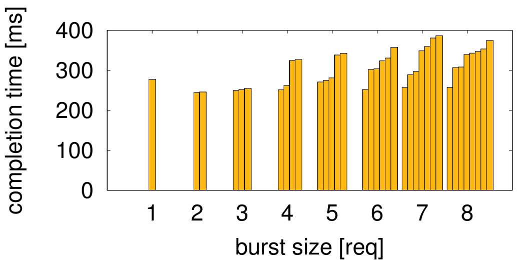
IV. SLA VIOLATION FORECASTING

In this section we present an analysis enabled by our approach - given a stream of arrivals and an SLA (i.e., a threshold over a percentile of the response time), can we predict if the system will violate it?

Our SLA violation forecasting is useful during the deployment of a new version of a system (e.g., hardware or software changes). The analysis attempts to answer the following question: given an SLA and a workload, will the system violate the SLA? We define the SLA by the tuple $< w^*, \alpha >$, where w^* is a threshold over the response time, and α is a response time



(a) sys1



(b) sys2

Fig. 3: Example of Burst Profiling Set for two systems (cut to burst size 8).

percentile. For such a tuple, the α-percentile of the response time has to be less than w^*. Since a working version of the system is already deployed, by simply feeding into the simulator the workload extracted by the logs of the current system and the burst performance set of the new system, we can predict if the new system violates the SLA. In the case of a true positive, the simulator correctly predicts that the system will handle the rate. A true negative means that the simulator accurately indicates that the system will violate the SLA. In a false negative situation, the simulator will judge the system not to be able to handle the workload, while in reality it can. Finally, in the case of a false positive, the simulator will incorrectly predict an SLA violation, when in fact the system is successfully able to sustain the rate. From a practical point of view, false positives would lead to dangerous situations where an under-sized system is wrongly judged to be well provisioned for a given workload. False negatives would lead to a waste of resources because the system is already able to meet the SLA, but the simulator suggests increasing capacity.

V. Evaluation

We evaluate our methodology along two dimensions. As a first step, we analyze the accuracy of the results provided by our methodology. Then, we evaluate the results provided by our approach for SLA forecasting analysis. We use a set of extensive experiments performed on two real systems as the baseline for our evaluation. The rest of this section is structured as follow: in Subsection V-A we describe the two systems we use for our experimental evaluation. Subsection V-B presents both the baseline used to experimentally validate our results and the burst performance set for the two systems. In subsection V-C we evaluate the accuracy of our methodology, and in Subsection V-D, we show the results of the SLA forecasting analysis we performed on the two systems.

A. Two systems

As the underlying architecture for both systems, we employ a 6 dual-core CPU machine equipped with 64GB of memory.

The first system, sys1, is an application server that offers a remote spell checking service. The server is implemented in Java using an event-driven architecture. The service can be queried through a Thrift interface. The algorithm used for the spell checker is similar to the one proposed in [18], and it uses a dictionary of about 150,000 English words.

The second system, sys2, is composed of several components and is more complex than sys1. The system runs an instance of the Mediawiki software running on a partial dump of the Wikipedia database. As a web server we used an Apache HTTP server version 2.4 with PHP version 5.5 as a script engine. To store the data we used a MySQL server version 5.5. In order to reduce the load on the database server, we configured Mediawiki to use a Memcached server. All the components described above have been installed in a single virtual machine. We employ KVM for the virtualization layer.

B. Baseline and profiling sets

We analyze both sys1 and sys2 with an extensive set of performance tests. Particularly, we first select different arrival rates ranging from low to high, and then we load the system with those rates to collect response time samples. Each rate is kept for at least 5 minutes. For sys1 we use 60 different rates in the range $[15, 206]\, req/s$, for sys2 we use 28 rates in the range $[1, 28]\, req/s$.

As explained in Section III, to predict the response time percentiles we use a pre-collected set of performance experiments. As a preliminary step, we create such a set for both systems. For sys1 we send bursts with sizes in the range $[1, 30]$, while for sys2 the range is $[1, 25]$. We send each burst 100 times.

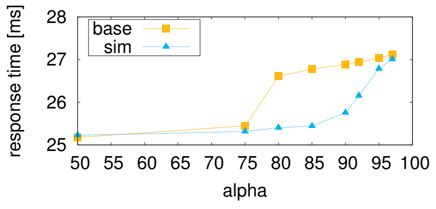
C. Accuracy

In this subsection we analyze the accuracy of the results provided by our methodology using the baseline presented in Subsection V-B. For each system, we select three arrival rates in order to analyze three different conditions: low, medium and high load. In particular, we select 16, 145 and 203 req/s for sys1, and 3, 11 and 25 req/s for sys2. Each rate is fed to the simulator, which produces a stream of response times (*sim*); the corresponding stream produced by the real system is named *base*. To evaluate the accuracy of the simulator we compare different levels of percentiles (α) of the response times generated by the simulator against the baseline. To quantitatively compare the two results, the relative error is computed as follows:

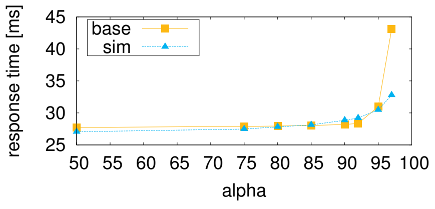
$$err = \left|\frac{pct(base, \alpha) - pct(sim, \alpha)}{pct(base, \alpha)}\right|$$

α	Low rate		Med rate		High rate	
	sys1 err [%]	sys2 err [%]	sys1 err [%]	sys2 err [%]	sys1 err [%]	sys2 err [%]
50	0.195	6.472	2.413	4.316	31.546	14.220
75	0.519	6.445	1.488	0.248	34.716	7.206
80	4.540	6.245	0.469	0.680	35.301	5.311
85	4.982	3.591	0.342	0.857	35.281	2.298
90	4.214	2.531	2.341	2.248	34.126	1.144
92	2.932	4.664	2.987	3.967	33.038	5.725
95	0.917	4.106	1.579	1.700	30.556	62.316
97	0.428	4.165	23.954	1.056	28.322	80.575

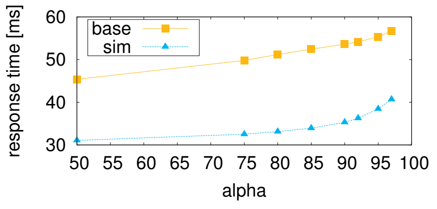
TABLE I: Accuracy of Burst Based Simulation for sys1 and sys2 under different loads.



(a) low load

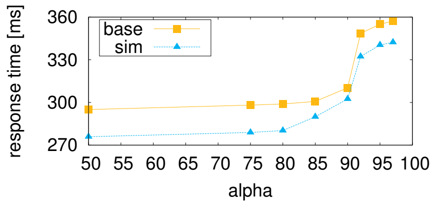


(b) med load

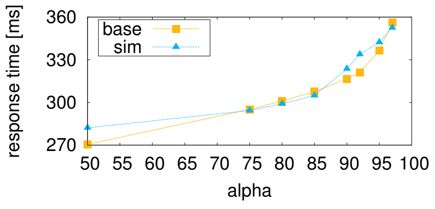


(c) high load

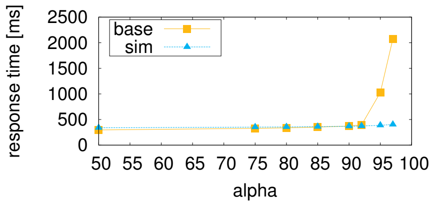
Fig. 4: Accuracy of Burst Based Simulation for sys1 under different loads.



(a) low load



(b) med load



(c) high load

Fig. 5: Accuracy of Burst Based Simulation for sys2 under different loads.

Table I summarizes the prediction error. Overall, the average error is 13.22 % for sys1 and 9.67 % for sys2. In particular, the prediction is very accurate for low and medium load, where the estimation error drops to 3.39 % for sys1 and 3.33 % for sys2. Instead, the performance of the estimation degrades significantly for the high load case: 32.86 % error for sys1 and 22.35 % for sys2. Estimating the percentiles of the response time for high-load cases is inherently difficult due to the unstable conditions of the system. Considering the percentiles, the estimation error rises above 80% for the 97^{th}-percentile of sys2 under high load. The estimation error rises significantly also for the 97^{th}-percentile of sys1 under medium load and for the 95^{th}-percentile of sys2 under high load. Other than the previous observations, there is no clear pattern in the relationship between α and the prediction error. In practice, however, the load of production systems is typically low [19], suggesting that being able to accurately predict the performance in these situations can be more valuable. Figures 4 and 5 show the percentile of the response time as a function of α both for the real system (base) and the simulator (sim). Unfortunately, the simulator tends to under-estimate the percentile of the response time (Figures 4a, 4c and 5a). This reduces the waste of resources allocated to a system, but could lead to dangerous conclusions. Finally, it is interesting to note that although the two systems present very different behaviors (see Figure 3), our methodology provides similar estimation errors for both.

D. Analysis 1 - SLA violation forecasting

In this subsection we present the results of the SLA violation forecasting analysis. We first define different SLAs for both systems. For sys1, we define all the possible combinations resulting from drawing w^* from the set { 25ms, 30ms, 35ms, 40ms, 45ms, 50ms, 55ms } and α from { 80, 85, 90, 95, 97 }. For sys2 we draw w^* from { 275ms, 290ms, 305ms, 320ms, 335ms, 350ms, 365ms, 380ms, 395ms } and α from the same set used for sys1. This results in 35 SLAs for sys1 and 45 for sys2.

Then, we select the six rates we used in Subsection V-C, three for each system. This results in 105 scenarios for sys1 and 135 for sys2. We analyze all the scenarios with the our simulator, and compare the outcomes with the baseline. Table II shows the number of true/false positives/negatives for both sys1 and sys2. We summarize these data with the classic metrics used in the literature:

$$accuracy = \frac{truepos + trueneg}{truepos + trueneg + falsepos + falseneg}$$

		sys1		**sys2**	
		Is rate sustainable?		**Is rate sustainable?**	
		yes	*no*	*yes*	*no*
Forecast	*yes*	59	20	59	7
	no	0	26	2	67

TABLE II: Result of the SLA violation forecasting analysis: 105 scenarios for sys1 and 135 for sys2.

$$precision = \frac{truepos}{truepos + falsepos}$$

The accuracy of the classification is 0.81 for sys1 and 0.93 for sys2. The precision is 0.75 and 0.89 for sys1 and sys2, respectively. Zooming into the raw data, we see that for 18 out of the 20 false positive of sys1 occur in case of high rates and 5 out of 7 false positive of sys2 occur in case of low rate. This can be explained with the findings reported in Subsection V-C: the simulator tends to under-estimate the percentile of the response time. Figures 4 and 5 show how the under-estimation is largest for the high load case for sys1 (Figure 4c) and the low load case for sys2 (Figure 5a).

VI. Related Work

Many studies in the literature already address the problem of the distribution of the response times. The general approach proposed in the literature is to model the behavior of real systems using complex queueing network models. For instance, the distribution of the response time is well known for $M/G/1/PS$ models [9, 8]. This family of models considers systems without parallelism and with a processor sharing discipline. Instead, other studies model the so-called "neighboring effect" of cloud platforms, considering systems where the computational capacity changes over the time due to exogenous factors [20, 11, 21, 10]. Serazzi et al. rely on simulation-based approaches to analyze complex queueing networks [22]. Other authors [5, 7] focus on the prediction of the average response time for multi-tier web applications. Watson et al. [6] predict the percentiles of the response time relying on little knowledge of the application. The novelty of our work lies in a completely automatic empirical-based approach that predicts the percentiles of the response time treating the whole system as black-box.

VII. Conclusion

In this paper, we address the critical and challenging question of how to predict SLAs associated with percentiles of response times for service systems. To capture complex system dynamics with a minimum overhead, we propose a two-phase methodology, which combines a novel burst profiling phase on real systems, and a state-dependent simulation model. Extensive evaluations on two types of bare-metal and virtualized service systems, show an average error rate of less than 15% when predicting a wide range of response time percentiles under various system loads. Moreover, based on the proposed methodology, we can accurately derive analysis on predicting SLA violations for different combinations of target values and response time percentiles.

VIII. Acknowledgments

The research presented in this paper has been supported by the Swiss National Science Foundation (project 200021_141002).

References

[1] M. Bjorkqvist, L. Y. Chen, and W. Binder, "Cost-driven Service Provisioning in Hybrid Clouds," in *Proceedings of*, ser. SOCA '12, pp. 1–8.
[2] M. Bjorkqvist, L. Chen, and W. Binder, "Opportunistic Service Provisioning in the Cloud," in *Proceedings of*, ser. CLOUD '12, June, pp. 237–244.
[3] J. Dean and L. A. Barroso, "The Tail at Scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[4] J. B. E Schurman, "The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search," in *Proceedings of Velocity, Web Performance and Operations Conference*, 2009.
[5] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An Analytical Model for Multi-tier Internet Services and Its Applications," in *Proceedings of*, ser. SIGMETRICS '05, pp. 291–302.
[6] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang, "Probabilistic Performance Modeling of Virtualized Resource Allocation," in *Proceedings of*, ser. ICAC '10, pp. 99–108.
[7] W. Iqbal, M. N. Dailey, and D. Carrera, "SLA-Driven Dynamic Resource Management for Multi-tier Web Applications in a Cloud," in *Proceedings of*, ser. CCGRID '10, pp. 832–837.
[8] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley, 1975.
[9] N. Gautam, *Analysis of Queues: Methods and Applications*, 1st ed. CRC Press, Inc., 2012.
[10] J.-P. Dorsman, M. Vlasiou, and B. Zwart, "Parallel Queueing Networks with Markov-modulated Service Speeds in Heavy Traffic," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 2, pp. 47–49, Aug. 2013.
[11] G. Casale and M. Tribastone, "Modelling Exogenous Variability in Cloud Deployments," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 73–82, Apr. 2013.
[12] L. Y. Chen, D. Ansaloni, E. Smirni, A. Yokokawa, and W. Binder, "Achieving Application-centric Performance Targets via Consolidation on Multicores: Myth or Reality?" in *Proceedings of*, ser. HPDC '12, pp. 37–48.
[13] A. Peternier, W. Binder, A. Yokokawa, and L. Chen, "Parallelism Profiling and Wall-time Prediction for Multi-threaded Applications," in *Proceedings of*, ser. ICPE '13, pp. 211–216.
[14] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "Dealing with Burstiness in Multi-Tier Applications: Models and Their Parameterization," *Software Engineering, IEEE Transactions on*, vol. 38, no. 5, pp. 1040–1053, Sept 2012.
[15] M. Welsh, D. Culler, and E. Brewer, "SEDA: An Architecture for Well-conditioned, Scalable Internet Services," in *Proceedings of*, ser. SOSP'01, 2001, pp. 230–243.
[16] S. Spicuglia, M. Björkqvist, L. Y. Chen, G. Serazzi, W. Binder, and E. Smirni, "On Load Balancing: a Mix-Aware Algorithm for Heterogeneous Systems," in *Proceedings of*, ser. ICPE'13, pp. 71–76.
[17] D. Ansaloni, L. Y. Chen, E. Smirni, and W. Binder, "Model-driven consolidation of java workloads on multicores," in *Proceedings of*, ser. DSN '12, pp. 37–38.
[18] W. A. Burkhard and R. M. Keller, "Some Approaches to Best-match File Searching," *Commun. ACM*, vol. 16, no. 4, pp. 230–236, Apr. 1973.
[19] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," in *Proceedings of*, ser. ASPLOS '14, pp. 127–144.
[20] S. R. Mahabhashyam and N. Gautam, "On Queues with Markov Modulated Service Rates," *Queueing Syst. Theory Appl.*, vol. 51, no. 1-2, pp. 89–113, Oct. 2005.
[21] B. Zhang and B. Zwart, "Fluid Models for Many-server Markovian Queues in a Changing Environment," *Operations Research Letters*, vol. 40, no. 6, pp. 573 – 577, 2012.
[22] M. Bertoli, G. Casale, and G. Serazzi, "JMT: Performance Engineering Tools for System Modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 10–15, 2009.