Response Time Characterization of Microservice-Based Systems

Jaime Correia, Fábio Ribeiro, Ricardo Filipe, Filipe Araujo and Jorge Cardoso
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
Emails: jaimec@dei.uc.pt, fdcr@student.dei.uc.pt, rafilipe@dei.uc.pt, filipius@uc.pt and jcardoso@dei.uc.pt

***Abstract*—In pursuit of faster development cycles, companies have favored small decoupled services over monoliths. Following this trend, distributed systems made of microservices have grown in scale and complexity, giving rise to a new set of operational problems. Even though this paradigm simplifies development, deployment, management of individual services, it hinders system observability. In particular, performance monitoring and analysis becomes more challenging, especially for critical production systems that have grown organically, operate continuously, and cannot afford the availability cost of online benchmarking. Additionally, these systems are often very large and expensive, thus being bad candidates for full-scale development replicas.**

Creating models of services and systems for characterization and formal analysis can alleviate the aforementioned issues. Since performance, namely response time, is the main interest of this work, we focused on bottleneck detection and optimal resource scheduling. We propose a method for modeling production services as queuing systems from request traces. Additionally, we provide analytical tools for response time characterization and optimal resource allocation. Our results show that a simple queuing system with a single queue and multiple homogeneous servers has a small parameter space that can be estimated in production. The resulting model can be used to accurately predict response time distribution and the necessary number of instances to maintain a desired service level, under a given load.

***Index Terms*—Modeling; Performance; Microservices; Observability; Tracing; Availability.**

I. Introduction

In dynamic, elastic production environments that scale in and out rapidly, tracking microservices performance can be a huge challenge. Black-box monitoring can be effective and light-weight, such as in our previous work [1]. However, the majority of these solutions lack an important feature: identifying and predicting quality of service. Despite their collected metrics, notifications and configurable dashboards, the burden of analysis rests on the administrators. We propose modeling components, as homogeneous multi-server queues, enabling a statistical characterization of fundamental performance metrics, such as latency and throughput. Moreover, since queues can be composed into networks, this approach can serve as the building block to model systems of microservices. To accurately model individual services, we need detailed measurements of their performance. Since we want to capture real workload conditions, we resort to tracing [2].

With the additional information given by tracing, we are able to create and update a dynamic dependency model for a system of microservices. This allows us to extract the dependencies between service endpoints, and, more importantly, we are able to model the performance of each microservice. The objective is simple, but very ambitious: obtain the optimal zone of operation for each service. This can lead to multiple pathways, such as understanding when to scale in or out, reducing the infrastructure cost, and ensuring the service level agreements (SLAs). Additionally, the possibility of pinpointing bottlenecks in the system without stressing it, is a major advantage for system administrators.

In this paper, we develop and instrument a microservice-based system, and resort to tracing, to model microservices as multi-server queues ($M/M/c$). This makes it possible to predict the distribution of response times and the optimal operation zone of a service, as well as determine how many instances would be needed to maintain the desired service level for a given workload. Moreover, having a performance model makes it possible to establish a notion of maximum capacity, and is the first step towards full system performance optimization and bottleneck detection. Our results demonstrate that although simple, our model can accurately predict the behavior of a microservice, more precisely, the response time distribution, without making the modeling and parameter estimation too complicated or too costly. Consequently, this type of model is adequate for online estimation and analysis.

The rest of the paper is organized as follows. Section II describes the queue-based model we use to characterize microservices. Section III describes the experimental setting. Section IV shows the results of our experiments. Section V evaluates the results. Section VI presents the related work. Section VII concludes the paper.

II. Performance Modeling

To analyze the performance of a system, and predictions its behavior, a suitable model is necessary. The detection of bottlenecks requires a notion of capacity limits, as a way to compare two services in an execution path, and to predict performance changes in response to resource allocation. In the context of microservice-based systems, resource allocation for scaling purposes is usually done at the instance level, with virtual machines or containers. If those requirements are fulfilled, it becomes possible to determine analytically which microservices have insufficient or excessive capacity.

978-1-5386-7659-2/18/$31.00

Our requirements for a model are the ability to represent response time, throughput and parallelism, as we intend to use it for online modeling. The analytical model needs to be composable in some way, to represents the resulting system.

Queueing systems are ideal for our use case, describing the aspects and metrics we are interested in, and being composable as queueing networks. It has been demonstrated that when a large number of customers independently make requests to a service, the arrival process will be Markovian [3]. We propose modeling microservices as $M/M/c$ queues.

Our parameter space is quite small and intuitive, $\{\lambda, \mu, c\}$. Here, λ and μ represent the rate parameters for the exponentially distributed inter-arrival and service times, and c the number of homogeneous servers, representing service parallelism. It is relevant to note that c does not necessarily represent the number of instances, as, in some cases, service instances will have some form of internal parallelism.

TABLE I
NOTATION

Symbol	Meaning
λ	Request arrival rate.
μ	Request service rate.
c	Number of parallel servers in the queueing system model.
I	Number of physical microservice instances.
ϕ	Internal parallelism of each instance.
W	Random variable representing time in queue.
Q	Random variable representing queue length.
S	Random variable representing service time.
T	Random variable representing total system time.
a_i	Arrival time of the i^{th} request.
w_i	Time spent in queue by the i^{th} request.
b_i	Service start time of the i^{th} request.
d_i	Departure time of the i^{th} request.
s_i	Service time of the i^{th} request.
t_i	Total system time of the i^{th} request.
ρ	Occupation rate.
Π_W	Probability that a request has to wait.

Table I presents the notation used in the rest of the paper. We have two direct applications for the model: 1) Estimate the maximum throughput capacity of an instance (bottleneck detection); 2) Determine the number of instances, to achieve a desired request time distribution, for a given request arrival rate (quality of service).

Given the $M/M/c$ model of a microservice of parameters $\{\widehat{\mu}, \widehat{c}\}$ we want to answer three questions. First, what is the maximum throughput capacity of the service? Second, under a given load λ, what is the distribution of total system time T? And finally, given a load λ, what is the necessary number of instances, to expect a percentage of requests below a system time threshold?

For the first question, an estimation of the maximum average throughput capacity of a $M/M/c$ queue is trivially obtained from $c\mu$. The other two questions can be answered by determining the Cumulative Distribution Function (CDF) $P(T < t)$ for a given queue of parameters $\{\lambda, \mu, c\}$, where T is the random variable representing total system time. As this formula is context dependent on the queue parameters $\{\lambda, \mu, c\}$, for clarity, we will sometimes use the notation $CDF(\lambda, \mu, c, t) = P(T < t)$, where T is the random variable for total system time of a request in an $M/M/c$ queue of parameters $\{\lambda, \mu, c\}$. Following Adan *et al.* [4], $P(T < t)$ can be calculated from W and S, the random variables representing waiting time and service time respectively, as shown in Equation 1.

$$
\begin{aligned}
&P(T < t) = 1 - P(W + S > t) \\
&= \begin{cases} 1 - \frac{\Pi_W}{1-c(1-\rho)} e^{-c\mu(1-\rho)t} + \left(1 - \frac{\Pi_W}{1-c(1-\rho)}\right) e^{-\mu t} & \text{if } c(1-\rho) \neq 1 \\ 1 - (\mu \Pi_W t + 1) e^{-\mu t} & \text{if } c(1-\rho) = 1 \end{cases}
\end{aligned} \tag{1}
$$

A natural application here, in the context of determining quality of service, is to compute the expected percentage of requests with total system time bigger than a given threshold r by computing $1 - P(T < r)$ for a predicted load.

For the last question, we wish to determine the smallest integer number of instances I that ensure (at least) a given distribution of total system time T, where the probability of a request time being over a threshold r, is at most a desired probability p, or $1 - P(T < r) \leq p$.

Parameter estimation: To model a given microservice, we need to estimate two parameters, the service rate $\widehat{\mu}$ and its parallelism $\widehat{c}$ from a sample of arrival and departure times for observed requests.

The estimation of request arrival rate $\widehat{\lambda}$ can be easily obtained using moment or maximum-likelihood methods. *For service rate* $\widehat{\mu}$, we need to make sure that the set of sampled requests ι was taken in an interval of time where the queue was empty, therefore $\sum_{i \in \iota} w_i = 0$. Finding an adequate interval may seem daunting at first, but once we have $\widehat{c}$, we can then look for a continuous set of samples $\mathbb{S}$ taken from an interval, where there are at most $\widehat{c}$ requests in the system. An alternative heuristic method, independent of $\widehat{c}$, is running a sliding window average of a statistically significant width, and finding the contiguous sub-sample with the lowest average, as we know that average service time monotonically increases with load (ρ).

To estimate the parallelism c, we used the method proposed by *Park et al.* [5], for analysis of unobservable queues. According to their method, the estimation problem can be expressed as the optimization statement in Equation 2, where $D_{(i;j)}$ is the ith order statistic, among the first j departure times.

$$
\begin{aligned}
\underset{\widehat{c}}{\text{Minimize}} \quad & \sum_{i=1}^{N} (\widehat{s_i} - E[\widehat{S}])^2 \\
\text{subject to} \quad & \widehat{b_i} = a_i, i = 1, \ldots, \widehat{c} \\
& \widehat{b_i} = \max\{a_i, D_{(i-\widehat{c};i-1)}\} \\
& \widehat{s_i} = d_i - \widehat{b_i}, \ \forall i \\
& \widehat{s_i} > 0, \ \forall i \\
& 1 \leq \widehat{c} \leq N, \ \widehat{c} \in \mathbb{N}
\end{aligned} \tag{2}
$$

Although accurate under simulation, this method is very sensitive to time measurement imprecisions, typical of real-life scenarios. These imprecisions can arise from parts of the workflow that are hard to measure, typically time spent in sockets, load balancers, in the operating system, and so on. As we already have a method for estimating the service rate $\widehat{\mu}$, as well as a sample of request times, we can calculate $\widehat{c}$ to maximize the Mean Square Error (MSE) between observed request time distribution $eCDF$ and the prediction CDF made with the model resulting from $\{\lambda, \widehat{\mu}, \widehat{c}\}$.

III. Experimental Setup

We validated our model choice and parameter estimation techniques against both a simulated and a real microservice. The simulation was used to validate the parametrization methodology, in a situation where the assumptions were known to hold true. The second experiment, with the real microservice was meant to validate both the quality of the parametrization in a real-world scenario, and measure the quality of the resulting model.

A. Simulation with `qcomputer`

Using `qcomputer` [6], an R package for queue system simulation, we create a simulated microservice, with exponentially distributed service times and a Poisson arrival process. Using this model, we generated samples of arrival and departure times for different parameterizations, from which we can then calculate response times. We then attempted to estimate the parameters $\{\lambda, \mu, c\}$ from the samples, using the estimators described in section II. Using the resulting model, we calculated the predicted response time distribution and compared it to the empirical distribution of the sample, to calculate the MSE between both.

B. Experiment with a Microservice

To test the approach in a more realistic setting, where some assumptions may not hold, we deployed a microservice-based application on a virtual machine with 16 vCores and 32GiB of system memory, running Ubuntu Server 16.04.3 LTS. The services, respectively, data stores and load balancers were packaged in docker containers and orchestrated and scaled using docker compose. The target microservice was behind a simple gateway service, and an `nginx` load balancer. Measurements of arrival and departure time, measured from the load balancer, were extracted using OpenTracing [7] compliant tracing instrumentation.

To load the deployments with approximately exponentially distributed inter-arrival request times, we used `Apache JMeter`. The naive approach would be to have each client thread sleeping for an exponentially distributed time between requests. However, since a client thread can only start a new request once it gets the previous reply, most time they will be left waiting for a reply, thus degenerating the distribution of inter-arrival times away from the desired exponential. To mitigate this effect, we attempted to simulate a large number of clients generating load at random intervals, which is known to produce approximately exponential inter-arrival times [3]. To do this, we ran the experiments with a number of threads much larger than the target load, and have them waiting a uniformly distributed random time between requests. The larger the number of threads, the larger the average waiting time, thus diluting the degeneration cause by the server response time. To achieve a load of L requests per second with τ threads, we generate the sleep time for threads from a random uniform number between 0 and U (milliseconds), according to $U = \frac{2000\tau}{L}$. Meaning each thread, on average, will make a request every $\frac{U}{2}$ milliseconds.

TABLE II
Estimation of μ from samples with different occupations (ρ).

ρ	c	μ	$\widehat{\mu}$	**Error (%)**
0.0333	1	30	29.24	3%
0.8333	1	30	5.02	83%
0.8333	2	30	9.88	67%
0.8333	3	30	2.13	93%
0.8889	3	30	8.50	72%

TABLE III
Parameter estimation results and error with simulation data.

Setting	$\widehat{\lambda}$	$\widehat{\mu}$	$\widehat{c}$	**MSE**	**Bias**
S1	0.99	29.24	1	0.000028	-0.00300
S2	24.89	29.24	1	0.001135	-0.02685
S3	50.38	29.24	2	0.000613	-0.01476
S4	74.92	29.24	3	0.001202	-0.02473
S5	79.43	29.24	3	0.002472	-0.04344

IV. Results

This sections presents the results obtained. To measure the error we calculated both MSE and Bias.

A. Simulation with `qcomputer`

We experimented with five settings in the simulation. The values $[1, 1, 2, 3, 3]$ for c, 30 for all μ and $[1, 25, 50, 75, 80]$ for λ, were used.

Using the sliding window average method, we obtained the results in Table II. Only the estimation in the first line is acceptable, and, in fact, very good. This shows the importance of selecting the correct sample for estimating service rate, as their quality will be inversely correlated with occupation. The best estimator for service rate $\widehat{\mu}$, it was used to model all the other settings, as they refer to the same simulated microservice. We know the load to be constant and the inter-arrival times exponentially distributed, for each setting, we got an estimate for arrival rate by the moments method.

When it came time to get an estimate of the parallelism of the service, $\widehat{c}$, we employed both Park *et al.* [5] and our alternative method to calculate it by minimizing the MSE. In Table III, presents the estimations obtained.

For each setting, we calculated the predicted cumulative distribution function $CDF(\widehat{\lambda}, \widehat{\mu}, \widehat{c}, t)$ and the empirical cumulative distribution function $eCDF(t)$ and calculated the MSE between samplings from those two functions. Figure 1 shows the $eCDF$ values, the prediction CDF and the cumulative

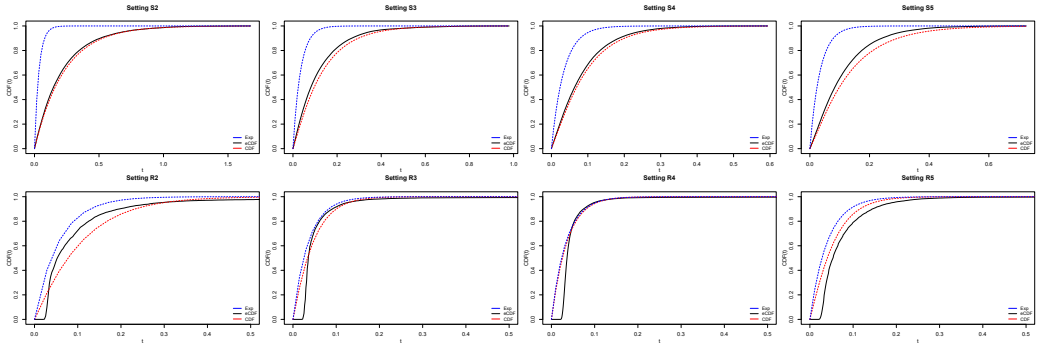

Fig. 1. Comparison of $CDF(\widehat{\lambda}, \widehat{\mu}, \widehat{c}, t)$ and $eCDF(t)$ for simulated settings and real settings.

TABLE IV
PARAMETER ESTIMATION RESULTS.

Setting	$\widehat{\lambda}$	$\widehat{\mu}$	$\widehat{c}$ (Park *et al.*)	$\widehat{c}$ (Optim.)
R1	16.16	17.50	1	2
R2	24.46	17.92	2	2
R3	48.86	28.05	3	3
R4	73.09	29.62	5	5
R5	90.91	24.91	7	5

TABLE V
PREDICTION MEAN SQUARE ERROR (MSE) AND BIAS.

Setting	Park *et al.*		Optimization	
	MSE	Bias	MSE	Bias
R1	0.082144	-0.195274	0.000704	0.014506
R2	0.000912	0.003089	0.000912	0.003089
R3	0.001453	0.007380	0.001453	0.007380
R4	0.003610	0.012120	0.003610	0.012120
R5	0.006047	0.030471	0.002498	0.019917

TABLE VI
PERFORMANCE PREDICTIONS.

Setting	ρ	$E[Q]$	$E[T]$
R1	0.46	0.25	0.073
R2	0.68	1.19	0.104
R3	0.58	0.46	0.045
R4	0.49	0.12	0.035
R5	0.73	1.16	0.053

distribution function of an exponential distribution of rate $\widehat{\mu}$ ("Exp" in the plots), which represents the distribution of service time (S). In simple terms, this would be the expected response time distribution if there was no queue time. The exact values are available in Table III. The MSE and Bias value demonstrate that $eCDF(t)$ and $CDF(\widehat{\lambda}, \widehat{\mu}, \widehat{c}, t)$ are quite close to each other, in the simulated environment. To conserve space, we omitted the figure for the best setting.

B. Experiment With a Microservice

We used five deployment settings to evaluate the parameterization technique and adequacy of the resulting model with the number of instances taking the values $[1, 1, 2, 3, 3]$ and $[16, 25, 50, 75, 85]$ for arrival rate. Given the arrival and departure times for each request, we estimated the model parameters. The service rate was estimated using the sliding window method, to maintain independence from $\widehat{c}$. Parallelism, $\widehat{c}$, was obtained using two methods, the one proposed by Park *et al*. [5], and the optimization that minimizes the MSE between observed and predicted response times. Table IV shows the results of the estimation step.

The second method (Optim.) produced better estimations and was used for the remainder of the experiment. Note however, that they could in principle be used together to get a higher quality estimation.

$CDF(\widehat{\lambda}, \widehat{\mu}, \widehat{c}, t)$ was calculated for each sample and compared to their $eCDF(t)$ (Figure 1). Table V shows the prediction error. Besides predicting the cumulative distribution function, we also calculated some other typical queueing system performance metrics, namely, mean queue length and mean request time. Table VI shows the resulting values, enabling a comparison between performance metrics and the occupation rate. Note that $E[T]$ increases as $\rho \to 1$ but it is not the only determining factor, $E[T]$ slowly decreases as c grows as the chance of finding an available server increases.

While the results still have acceptable error, they not as good as the ones from the simulated experiment. This observation is explored in detail in Section V.

V. DISCUSSION

The experiments with the simulated data exposed two important aspects. First, the quality of the estimation of service rate ($\widehat{\mu}$) is very dependent on occupation. To get a good estimate of service rate, we need to sample it under low occupation. With setting *S1*, we were able to get an estimate $\widehat{\mu}$, and accurately predict the behavior of the other settings.

In a real setting there are measurement imprecisions and external factors. In this particular case, we noticed that due to some implementation detail, the service time was not independent from the number of instances, and improved with the number of instances (refer to Table IV). This might be an artifact of some caching at the back-end, in this case a shared data store, which is a distortion of the intended setting. So far, we avoided more general queuing systems ($G/G/c$), as they are not easily composable and lack closed solutions.

There are some limitations inherent to the approach. The estimation method for service rate is very sensitive to occupa-

tion, thus requiring a favorable sampling period, where load on the microservice to be modeled is low. The estimation methods for parallelism have distinct limitations. The one proposed by Park *et al*. [5] requires having complete sampling, starting at an instant when there were no requests in the system The optimization method, requires already having a good estimate of service rate, and will be dependent on its quality.

As for the model itself, since $E[T]$ grows asymptotically to infinity as $\rho \to 1$, the model is far more sensitive to error in estimation, under higher loads, for which we are attempting to predict the distribution of T.

Our goal for the future is composing the individual models in networks. A deeper characterization and further study of this modeling approach is necessary. As such, we will run additional experiments in a more varied set of instrumented microservices. To model services in place in a production setting, methods to quantify the impact of services on other upstream services, which depend on them, will have to be developed. For services that have more general service rates, we will attempt to model them as tandem queues.

VI. Related Work

The literature is rich with approaches to model and analyze systems. Bahl *et al*. [8], uses network traffic to create and inference Graph model, to check service degradation and failures in an enterprise network. However, it needs the enterprise network topology, and therefore it is closely coupled to the system. Urgaonkar *et al*. [9], propose an analytically model, for multi-tier internet services, using multi-tier queue, where each queue representing different layers of the application. Similarly, Bi *et al*. [10], use $M/M/c$ queues in an open network to model multi-tier systems for dynamic provisioning in cloud deployments. Although associate with our approach, they aim to model multi-tier services, while our goal is to create model of microservice-based distributed systems. Using networks of queues or layered queues is suggested in various works [11]–[13]. However they focus on manual modeling at design time or from deep knowledge of the system, instead of trying to extract them from an existing system or considering other data driven approaches. There are approaches to model the performance and response times of services, [14] suggests modeling classic web servers as $M/G/1/K * PS$ queues.

While others have explored similar approaches, models with no assumption about processing time, like $M/G/1$, $G/G/c$, although more expressive have no closed analytical solutions and cannot be easily composed. Heinrich *et al*. [15] explores the intricacies of microservice-based systems and lays out the existing challenges. He points out that existing modeling approaches are not adequate for the typical scenarios of the modern microservice-based systems.

VII. Conclusion

In this paper we proposed a queuing theory based modeling approach to characterize the performance of microservice-based systems. Models of individual microservices can be used to characterize their response time distribution. The accompanying analytical tools can be used to predict response times under any load and determine the required number of instances to maintain a desired quality of service. The individual models are composable, for example as networks, and can in the future be used to reason about the performance of the resulting system. In that setting, as the model gives an implicit notion of capacity, they can additionally be used for bottleneck detection and optimal resource allocation at a global level.

Acknowledgments

This work was carried out under the project PTDC/EEI-ESS/1189/2014 — Data Science for Non- Programmers, supported by COMPETE 2020, Portugal 2020- POCI, UE-FEDER and FCT. We would also like to express our gratitude to the INCD - *Infraestrutura Nacional de Computação Distribuída*, for providing access to their computational resources.

References

[1] R. P. R. Filipe and F. Araujo, "Client-side black-box monitoring for web sites," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, Oct 2017.

[2] R. R. Sambasivan, I. Shafer, J. Mace, B. H. Sigelman, R. Fonseca, and G. R. Ganger, "Principled workflow-centric tracing of distributed systems," in *Proceedings of the Seventh ACM Symposium on Cloud Computing - SoCC '16*. New York, New York, USA: ACM Press, 2016, pp. 401–414.

[3] A. A. Shahin, "Enhancing Elasticity of SaaS Applications using Queuing Theory," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 8, no. 1, pp. 279–285, 2017.

[4] I. Adan and J. Resing, "Queueing theory," 2015.

[5] J. Park, Y. B. Kim, and T. R. Willemain, "Analysis of an unobservable queue using arrival and departure times," *Computers and Industrial Engineering*, vol. 61, no. 3, pp. 842–847, 2011.

[6] A. Ebert, P. Wu, K. Mengersen, and F. Ruggeri, "Computationally Efficient Simulation of Queues: The R Package queuecomputer," mar 2017.

[7] "Opentracing," http://opentracing.io/, retrieved Oct, 2017.

[8] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 13–24, Aug. 2007.

[9] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 291–302, Jun. 2005.

[10] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010*. IEEE, jul 2010, pp. 370–377.

[11] H. Li, "A Queue Theory Based Response Time Model for Web Services Chain," *2010 International Conference on Computational Intelligence and Software Engineering*, pp. 1–4, 2010.

[12] W.-p. Yang, L.-c. Wang, and H.-p. Wen, "A queueing analytical model for service mashup in mobile cloud computing," *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2096–2101, apr 2013.

[13] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, "Web server performance measurement and modeling techniques," *Performance Evaluation*, vol. 33, no. 1, pp. 5–26, jun 1998.

[14] J. Cao, M. Andersson, C. Nyberg, and M. Kihl, "Web Server Performance Modeling using an M/G/1/K*PS Queue," in *10th International Conference on Telecommunications, ICT 2003*, vol. 2, no. 2, 2003, pp. 1501–1506.

[15] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance Engineering for Microservices," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion - ICPE '17 Companion*. New York, New York, USA: ACM Press, 2017, pp. 223–226.