Towards Occupation Inference in Non-instrumented Services

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

***Abstract*—Measuring the capacity and modeling the response to load of a real distributed system and its components requires painstaking instrumentation. Even though it greatly improves observability, instrumentation may not be desirable, due to cost, or possible due to legacy constraints.**

To model how a component responds to load and estimate its maximum capacity, and in turn act in time to preserve quality of service, we need a way to measure component occupation. Hence, recovering the occupation of internal non-instrumented components is extremely useful for system operators, as they need to ensure responsiveness of each one of these components and ways to plan resource provisioning. Unfortunately, complex systems will often exhibit non-linear responses that resist any simple closed-form decomposition.

To achieve this decomposition in small subsets of non-instrumented components, we propose training a neural network that computes their respective occupations. We consider a subsystem comprised of two simple sequential components and resort to simulation, to evaluate the neural network against an optimal baseline solution.

Results show that our approach can indeed infer the occupation of the layers with high accuracy, thus showing that the sampled distribution preserves enough information about the components. Hence, neural networks can improve the observability of online distributed systems in parts that lack instrumentation.

***Index Terms*—Monitoring, Observability, Black-Box, Analytics, Neural Networks, Deep Learning, Performance Modeling**

I. Introduction

Observability – a metric of how well the internal state a sytem can be determined from its external outputs [1] – is key to ensure responsiveness of large-scale online systems, comprised of fine-grained distributed components, like microservices. Perfect observability would require extensive instrumentation of source code with agents dedicated to software and hardware resources. Additionally heavyweight systems, would be required to gather, store, process and display data in dashboards.

Unfortunately, an intrusive monitoring solution may not be desirable or possible due to technical constraints. This can happen due to resources in third-party providers (e.g., Content Delivery Networks), or lack of instrumentation, as this might be too complex or too expensive to cover the entire system.

Hence, inferring occupation of individual components without help from instrumentation or external agents can bring concrete benefits for the observability of the system. A clear use case, would be extracting information at sub-instrumentation granularity as well as improving the visibility over legacy parts of a system that are not or cannot be adequately instrumented. Given this goal, we aim to determine whether we can perform such separation using a neural network.

To evaluate this possibility, we created a laboratory experiment, where we use a system with two sequential $M/M/1$ queues. We opted for Markovian queue systems since request rates for modern use cases are known to be well modeled by Poisson processes for large numbers of clients [2]. Furthermore, as it has been shown that any system can be decomposed into an arbitrary number of queues as a result of the properties of sums of Markovian processes [3], this is relatively representative of software components. The intuition behind this property, is that computers, as discrete systems, can be thought of as a network of buffers.

We ran a set of several combinations for layer occupations, from lightly occupied to heavily busy. For each combination, we collected the response time, for a batch of client requests. Using this data, we trained a neural network, which we eventually set to three hidden layers of 100 neurons each, with two outputs representing the level of occupation of each component of the system. The point is to understand if the neural network could predict the occupation of each layer without expert understanding of the system. We evaluated our trained neural network against a baseline optimization method. To extract the occupation, this method explicitly uses underlying knowledge of the components, to fit the observed data with a tandem queue model. The aforementioned premises will be clarified in the following Sections.

Our experiments show that the neural network can accurately infer the occupation of each layer. With the exception of the case where one of the layers is extremely busy and dominates the response time of the system, both methods, the neural network and the tandem queue model, achieve satisfactory results. These results show the feasibility of using machine learning to do black-box monitoring of parts of the system with little or no observability. Furthermore, it reinforces that the measurements contain enough information to reason about the structure of the system that generated it.

The rest of the paper is organized as follows. Section II describes the methods we used for the problem we tackle in this paper. Section III describes the settings for our experiment.

978-1-7281-2522-0/19/$31.00

In Section IV we show and evaluate the meaning of the results, the strengths of this approach and its limitations for both methods. Section V presents the related work. Section VI concludes the paper and describes future directions.

II. System and Machine Learning Approach

In this Section, we describe our approach to infer the occupation of each component in the system of two queues depicted in Figure 1.

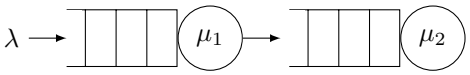


Fig. 1: Tandem $M/M/1$ queues.

Using a sample of response times, we determine the occupation of each component (which we refer to as layer), both with model fitting and a deep neural network. While the model fitting algorithm explores the underlying structure of the system and serves to prove that it is indeed possible to do the black-box prediction, our goal is to create a neural network that precludes the need for any assumption or knowledge about the system.

A. Tandem Queue Model Fitting

To determine the occupation of each layer of a tandem, two-component system, we first need to model their response to load. In a generic way, this response function is defined by the time it takes to service each request and the level of parallelism. Since we know the response time is the sum of the two random processes representing the service time portions happening at each layer, as a naive approach, we could attempt to model the data, by fitting the sum of two random variables (exponential for example). This would shed light on the time spent on each layer, and indirectly their capacity. However, this approach fails to capture the variation in response time in response to load/occupation, as a result of not considering the time spent waiting for the service.

Queuing theory gives us a theoretical framework to predict response time variation in function of occupation. As such, we modelled the system as a network of two tandem single-server queues ($M/M/1$), shown in Figure 1. This model assumes Markovian properties for both inter-arrival times as well as service times. It further assumes no parallelism. We forewent more general models, for which approximate or numerical solutions are known, as the objective is not generally solving the problem, but to prove its feasibility and establish a performance baseline for a relatively simple case. Each queue is defined by its arrival rate λ and service time μ, and the occupation ρ is $\frac{\lambda}{\mu}$. The probability density function (*PDF*) for the response time distribution is given in Equation 1 by $r(\lambda, \mu, x)$ for all values x in its support.

The model resulting of the composition of two tandem $M/M/1$ queues is defined by the arrival rate (λ), and the service rate of each queue (μ_1, μ_2), and has a response time distribution given in Equation 2 as $t(\lambda, \mu_1, \mu_2, x)$ and a respective cumulative distribution function (*CDF*) given in Equation 3 as $T(\lambda, \mu_1, \mu_2, x)$. Note that due to space restrictions, the numerator on $\tau(\lambda, \mu_1, \mu_2, x)$ is split in two lines.

As we want to find the occupations ρ of each layer, and $\rho \in]0, 1[$, we rewrite the model in terms of occupation, as shown in Equation 5.

To fit it to the data, we use optimization to find the values ρ_1, ρ_2 that minimize the mean square error (*MSE*) between Θ and the empirical cumulative distribution function (*eCDF*) of the samples. We assume λ is known for the time interval when the samples where taken. Figure 2 shows an example of how the model Θ fits the *eCDF* after the optmization step. This particular sample was generated from a system with a $(0.2, 0.7)$ occupation, and the optimization determined parameters $(0.16, 0.73)$.

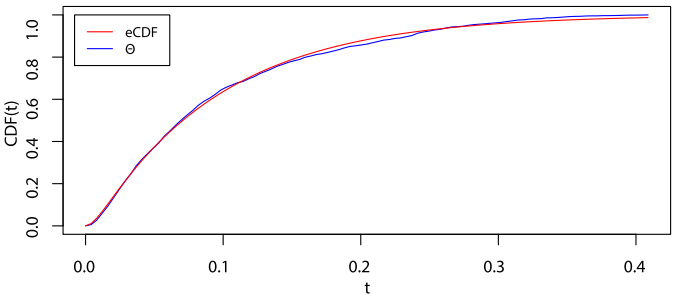


Fig. 2: Empirical and predicted Cumulative Distribution Functions (*CDF*).

B. Machine Learning Approach

We used a neural network that predicts each system layer's occupation from raw response times, as clients or components upstream observe them. Since we wanted to predict the output of a continuous value, our neural network would have to solve a regression problem, i.e., we want to predict our output value as accurately as possible — contrasting with a classification problem.

We made several tests with distinct algorithms and opted for a deep neural network. The rational being that we wanted to correlate both layers' occupation, since the output visible to the client is associated with both layers and has a complex non-linear relation. Furthermore, current software frameworks make deployment and use simple, as well as production-ready. In addition, we experimented several setups for the neural network — distinct number of layers, nodes (neurons), and activation functions. Our final configuration for the neural network consisted of one input layer with 2000 nodes, three hidden layers, each with 100 nodes, with the activation function being the `relu` function [4] and, in the final layer, a linear activation function. Since the network is shared by both outputs, we were able to have a multi-output regression model to predict each layer's occupation. Hence, both occupations are correlated and influence the hidden layers, having an impact

$$r(\lambda,\mu,x) = \begin{cases} (1-\rho)\mu e^{-x(1-\rho)\mu} & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\begin{aligned} t(\lambda,\mu_1,\mu_2,x) &= (r * r)(x) \\ &= \int_0^x r(\lambda,\mu_1,z) r(\lambda,\mu_2,x-z)\ dz \\ &= \mu_1\mu_2 \left(1-\frac{\lambda}{\mu_1}\right)\left(1-\frac{\lambda}{\mu_2}\right)\left(\frac{e^{x\lambda-\mu_1 x}}{\mu_2-\mu_1} - \frac{e^{x\lambda-\mu_2 x}}{\mu_2-\mu_1}\right) \end{aligned} \tag{2}$$

$$\begin{aligned} T(\lambda,\mu_1,\mu_2,x) &= \int_0^x t(\lambda,\mu_1,\mu_2,z)\ dz \\ &= \begin{cases} \left(1-\frac{\lambda}{\mu_1}\right)^2 {\mu_1}^2 \left(\frac{1-((\mu_1-\lambda)x+1)e^{\lambda x-\mu_1 x}}{{\mu_1}^2-2\lambda\mu_1+\lambda^2}\right) & \mu_1 = \mu_2 \\ \tau(\lambda,\mu_1,\mu_2,x) & \text{otherwise} \end{cases} \end{aligned} \tag{3}$$

where,

$$\tau(\lambda,\mu_1,\mu_2,x) = \frac{\left[\begin{array}{l} \left(1-\frac{\lambda}{\mu_1}\right)\mu_1\left(1-\frac{\lambda}{\mu_2}\right)\mu_2 e^{-\mu_2 x-\mu_1 x} \\ \quad \left((\mu_1-\lambda)\, e^{\mu_1 x+\lambda x} + \left((\mu_2-\mu_1)e^{\mu_1 x} + (\lambda-\mu_2)\, e^{\lambda x}\right) e^{\mu_2 x}\right) \end{array}\right]}{(\mu_1-\lambda){\mu_2}^2 + (\lambda^2-{\mu_1}^2)\mu_2 + \lambda{\mu_1}^2 - \lambda^2\mu_1} \tag{4}$$

$$\Theta(\lambda,\rho_1,\rho_2,x) = T(\lambda, \frac{\mu_1}{\lambda}, \frac{\mu_2}{\lambda}, x) \tag{5}$$

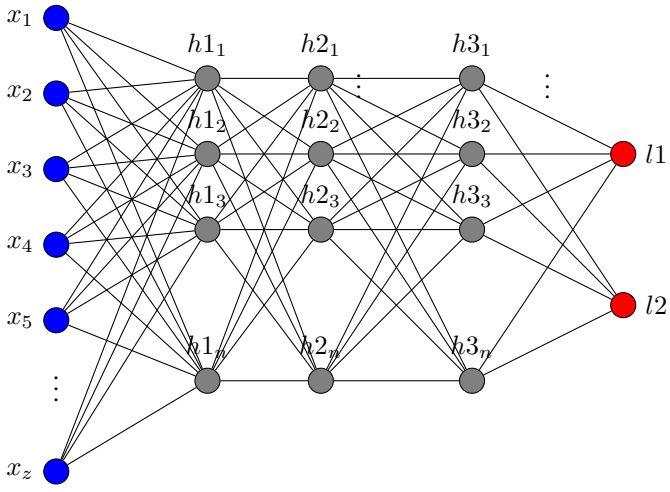


Fig. 3: Representation of the network for the two components

on the outcome. Figure 3 illustrates our model. The network receives z input values, which are the response times seen by z client requests, and outputs, l_i corresponding to the occupation of the ith layer.

We provide to our network around $10,000$ lines of raw data from our experiment. Since we are training our neural network, and therefore using a supervised machine learning approach, we need labeled data. Each line has $2,002$ values: $2,000$ response times as seen by clients and the labels: the $2,001^{st}$ value is the occupation of one layer, the $2,002^{nd}$ value is the occupation of the other layer. In the test scenario this output is not available, serving only to validate the accuracy of the prediction.

III. Evaluation

To validate the tandem queue model fitting method and the neural network, we used response time data generated with a simulated two-component system. We modeled a two component system as two sequential single server queueing components since it elegantly expresses the variation of response time with occupation. The simulation was made using the `qcomputer` [5] package written in `R`.

The occupation levels were defined as all the combinations of 0.1 increments in the range $[0.1, 0.9]$ - e.g., Layer 1 at 0.1 and Layer 2 at 0.5. The arrival rate was fixed at 30 requests per unit of time and the service rate of each layer varied to express the desired occupation. For each combination of occupations, we collected 150 samples. Since we had 9 occupation levels per component, we collected $9^2 * 150 = 12150$ samples of $2,000$ request observations each. These observations correspond to overall response times of each request.

To evaluate and create the neural network model, we used `Tensorflow` with `Keras` [4]. This framework, allowed us to rapidly generate and save our model and is a common standard for the generation of complex networks in the industry and academy. Of the 150 samples, 80%, or 120 per occupation combination, were used to train the neural network, and the remaining (30 per combination) for the validation of

both methods — test set. Furthermore, of the portion allocated to the neural network training, 30% (30% of the 80% portion) were used for model validation in `Tensorflow`.

The neural network treated the $2,000$ request observations as its input features and outputted the two occupation values. We used a `Sequential` model and the Mean Absolute Error — for the optimization score function —, and a total of 100 iterations over the dataset — i.e., "epochs".

For validation and comparison of the two approaches (neural network and model fitting), we calculated the following error metrics: Mean Square Error (MSE), Mean Absolute Error (MAE), as well as the mean Euclidean distance, because we wanted to understand the distance between the two-dimensional predictions. These metrics were calculated for the whole set of predictions, by range for each layer occupation as well as for each combination of occupations (Euclidean distance). MSE is particularly useful to compare our current results with the results of the methods that inspired this work [6].

IV. Results

Having generated predictions for a test set of $2,430$ samples (30 samples for each of the 81 combinations of occupations), we calculated their respective errors. Besides the global and per range error for each individual layer predictor, for which we used MSE and MAE, we calculated the error as the Euclidean distance for the prediction pair. This latter metric gives an absolute error value that more intuitively shows the quality of the prediction and better exposes issues such as the prediction regressing to the mean of the two occupation values.

We focus on the MAE and mean Euclidean distance since they have the same unit. Table I shows the MAE and MSE for each layer, as well as the respective standard deviations. The Mean Euclidean distances for the Tandem Queue Model Fitting and the Deep Neural Network are 0.09 ± 0.10 and 0.15 ± 0.12, respectively. Euclidean distances are further detailed in Figures 4c and 4d.

The best method in our previous work [6], has a MSE of 0.05 ± 0.04 and 0.05 ± 0.03, for layer 1 and 2 respectively. The new methods both show significant improvement over those same metrics, as shown on Table I. Moreover, both methods show improvement over all ranges of occupation. Table II shows error metrics for each method and layer, grouped by range. For example $\rho = 0.1$ shows the results for the pairs $(0.1, *)$ and $(*, 0.1)$, where the asterisk stands for all values. As neither model could distinguish the order of the occupation values — $(0.1, 0.5)$ is indistinguishable from $(0.5, 0.1)$ — the errors were calculated after sorting the layer values. The results are much more consistent over the whole range, compared to our previous work [6], where there was clear degradation, especially on the model fitting approach.

However, the error of each individual layer does not convey an important aspect of the prediction quality. We wish to preserve the relationship between the two occupation values, meaning that $(0.1, 0.5)$ should be predicted as such, instead of averaging the values to $(0.3, 0.3)$. To understand if this relationships exists, we measure the error as the Euclidean distance between the target and prediction pairs. Figure 4 shows the error in a way that preserves that relation. We present two visualizations for each method.

Firstly, we show the predictions as a cloud of points color coded by expected occupation pair (Figures 4a and 4b). The true occupations have a black circumference; predictions are dots with the same color as the disk inside the circumference. For example, in Figure 4a, we see that for occupations $(0.1, 0.1)$ all the predictions are clustered around the real value. As occupation increases, they get more disperse. Due to the optimization method used, we observe some values getting clamped creating a line in the diagonal. On Figure 4b, representing the neural network results, we see there is a particular pattern of dispersion for the extremes. On high occupations, predictions are shifted along one of the axes. This means that the neural network is able to accurately predict the higher occupation value, but the second one will tend towards central values. This is expected, since the highest occupied component dominates the overall response time experimented by the client.

Secondly, we show bubble charts, where the radius of each bubble is the mean Euclidean distance of the predictions from the real value. Here it is visible that the queue model, in Figure 4c, shows stability along the range, except on the highest occupation values. Figure 4d, relative to the Neural Network, presents a different pattern, having lower accuracy when the difference in occupations is highest ($(0.1, 0.9)$) or as it get closer to $(0.9, 0.9)$.

Results show that we can estimate the internal occupations from the system's response time with small error. When at least one of the components is very busy, only one of the high occupations is correctly approximated. We do not regard this as problematic, because a very busy component will dominate the response time anyway.

V. Related Work

We divide previous related work into three parts: monitoring tools and methods to gather server-side data, tracing frameworks, and modeling of computational machines. Traditional monitoring and tracing were studied as the standard alternatives to the approach we propose.

A. Monitoring tools

Traditional monitoring tools, like Nagios [7] or Zabbix [8], use probes or agents to collect infrastructure and application metrics, such as response time, load, and other sorts of numbers and status. Application Performance Monitoring (APM) solutions, like New Relic [9] or Dynatrace [10], go a little bit deeper and can use specific agents to automatically extract information about the internal components of distributed applications, e.g., a database. Other approaches, take advantage of architectural patterns and extract data at the platform level,

TABLE I: Global results for both methods.

Method	Layer 1		Layer 2	
	MAE	MSE	MAE	MSE
Queue Model	0.05 ± 0.07	0.01 ± 0.03	0.05 ± 0.08	0.01 ± 0.03
Neural Network	0.09 ± 0.11	0.02 ± 0.04	0.08 ± 0.10	0.02 ± 0.04

TABLE II: Error metrics for each method and layer, grouped by range.

	Queue model				Neural network			
	Layer 1		Layer 2		Layer 1		Layer 2	
ρ	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
0.1	$.05 \pm .09$	$.01 \pm .04$	$.05 \pm .09$	$.01 \pm .04$	$.12 \pm .14$	$.04 \pm .06$	$.11 \pm .13$	$.03 \pm .05$
0.2	$.05 \pm .08$	$.01 \pm .03$	$.06 \pm .11$	$.02 \pm .05$	$.1 \pm .1$	$.02 \pm .04$	$.11 \pm .11$	$.02 \pm .04$
0.3	$.06 \pm .09$	$.01 \pm .03$	$.06 \pm .09$	$.01 \pm .04$	$.1 \pm .08$	$.02 \pm .02$	$.08 \pm .07$	$.01 \pm .02$
0.4	$.06 \pm .08$	$.01 \pm .03$	$.07 \pm .09$	$.01 \pm .03$	$.09 \pm .08$	$.02 \pm .02$	$.08 \pm .07$	$.01 \pm .02$
0.5	$.06 \pm .07$	$.01 \pm .02$	$.06 \pm .08$	$.01 \pm .03$	$.09 \pm .09$	$.02 \pm .03$	$.08 \pm .08$	$.01 \pm .03$
0.6	$.06 \pm .07$	$.01 \pm .03$	$.05 \pm .07$	$.01 \pm .03$	$.09 \pm .1$	$.02 \pm .04$	$.09 \pm .1$	$.02 \pm .04$
0.7	$.05 \pm .05$	$.01 \pm .02$	$.05 \pm .06$	$.01 \pm .03$	$.1 \pm .1$	$.02 \pm .04$	$.09 \pm .1$	$.02 \pm .05$
0.8	$.04 \pm .05$	$0 \pm .03$	$.04 \pm .05$	$0 \pm .2$	$.1 \pm .13$	$.03 \pm .06$	$.08 \pm .11$	$.02 \pm .05$
0.9	$.04 \pm .04$	$0 \pm .01$	$.04 \pm .03$	0 ± 0	$.06 \pm .13$	$.02 \pm .06$	$.01 \pm .02$	0 ± 0

such as Pina *et al.* [11]. Eventually, all this data ends up in dashboards for the system administrators. Cloud providers are also an interesting case to cover here, as they have their own monitoring tools, like Amazon CloudWatch [12] or Azure Monitor [13].

B. Tracing methodologies

Some of the previous tools can even support distributed tracing. Tracing, unlike standard monitoring solutions, exposes causality relationships in the logs, allowing users to make inferences concerning critical paths and relations, e.g., among microservices. Regarding tracing, there are two major fields-of-work: black-box and non-black-box approaches. Concerning black-box methodologies, Aguilera *et al.* [14] used a tool that tracks message-level traces of the system, to debug the overall distributed system (not performing overall performance diagnosis). Tak *et al.* [15] use threads and network activities, as a middleware to detect request paths.

Relative to non-black-box, Sigelman *et al.* [16] created a tracing infrastructure for infrastructure and distributed applications. Sambasivan *et al.* [17] use the approach to gain insight at the application level, in particular workflow-based tracing, concerning the tracing of individual requests.

Linkedin [18] has a distributed tracing system built upon Apache Samza [19], to detect performance issues and root cause analysis. The system uses the call paths aggregation of every 15 minutes. Netflix [20] has also created several monitoring tools with distributed tracing as well as failure injection features, to improve resilience of the overall infrastructure.

OpenTracing [21] gives developers tracing clients in multiple languages and brings integration with the state-of-the-art tracing back-end tools. Google recently published a competing standard, OpenCensus [22], supporting a partially overlapping set of the same back-end tracing tools.

C. Performance modeling techniques

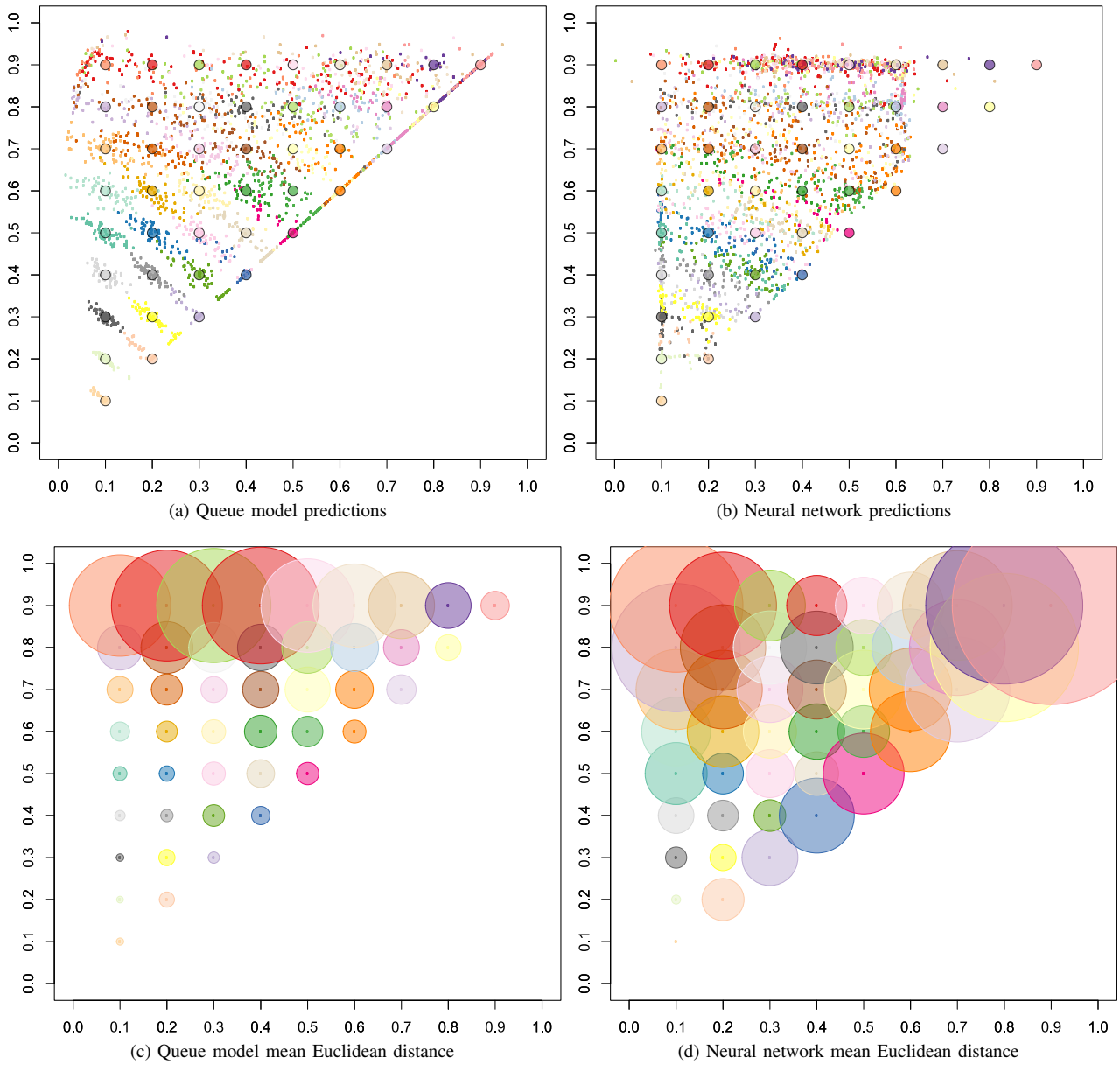
In [23], authors present a survey concerning microservice monitoring design and possible implementations, to promote monitoring standards. We have worked on performance modeling and monitoring before as well, e.g., in [24] or [6]. In [24], we used a black-box technique to detect internal and external bottlenecks of the system, using only client-side data and machine learning techniques. In [6] we also used a black-box approach for multi-component servers, but we now improve those results.

In the literature we can find plenty of approaches to model and analyze system performance. Bahl *et al.* [25] create an inference Graph model from network traffic to check service degradation and failures. However, their work is strongly tied to the enterprise network topology. Urgaonkar *et al.* [26], propose an analytical model based on multi-tier queues for multi-tier Internet services. Despite being related to our own work, Urgaonkar *et al.* model a multi-tier server, while we aim to do black-box monitoring. Other works, like [27]–[29] use networks or layered queues. However they do manual modeling at design time or modeling from expert knowledge of the system, instead of trying to extract the model from an existing system in a data driven fashion. Other approaches try to model the service performance and response times. This is the case of Cao *et al.* [30], who model classic web servers as $M/G/1/K * PS$ queues. However, models like $M/G/1$, $G/G/c$, with no assumption of processing time, are not amenable to closed-form solutions and cannot be easily composed.

Heinrich *et al.* [31] explore microservice-based systems and point out that the modeling approaches available do not fit modern microservice-based systems.

VI. Conclusion

Monitoring of highly distributed, dynamic, elastic systems is a herculean task for administrators, operators and even for developers. New software releases, agents, tracing, and a plethora of system monitoring tools and dashboards creates a complex environment for administrators to ensure correctness and proper quality-of-service.



(a) Queue model predictions

(b) Neural network predictions

(c) Queue model mean Euclidean distance

(d) Neural network mean Euclidean distance

Fig. 4: Tandem queue and neural network regression models error visualizations. Axes represent the occupation of each component.

In this paper we explored the possibility of inferring system occupation from the client point-of-view — a scenario ideal for legacy systems or where it is too complex or infeasible to instrument a small subset of components. Once occupation can be predicted, the system can react and leverage the elasticity of the supporting cloud platform to maintain the desired throughput and quality of service. This may be feasible even for more current state-of-the-art methodologies, such as microservices, where a module may be responsible to create new containers reacting to a lower quality-of-service.

Our objective was to identify occupation for a two-layer subsystem. More specifically, we wanted to compare two distinct methods: first, an optimized algorithm specifically designed to our scenario, and secondly, a neural network trained with the data collected from our experiment. Our results show that it is viable to infer the load of each layer collecting only the overall response time. Hence, these two methodologies – neural network and tandem queue model – are able to improve current monitoring tools, and ensure a more fine-grained knowledge about the system.

For future work, we intend to study and categorize real world response time data, using that knowledge to extend this methodology for more generic topological inference. In particular, we are interested in the number of components,

parallelism and occupation.

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.

REFERENCES

[1] R. Kalman, "On the general theory of control systems," *IRE Transactions on Automatic Control*, vol. 4, no. 3, pp. 110–110, dec 1959.

[2] 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. [Online]. Available: www.ijacsa.thesai.org

[3] A. Horváth and M. Telek, "Phfit: A general phase-type fitting tool," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 2002, pp. 82–91.

[4] "Tensorflow," https://www.tensorflow.org/guide/keras, retrieved Feb, 2019.

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

[6] R. Filipe, J. Correia, F. Araujo, and J. Cardoso, "On black-box monitoring techniques for multi-component services," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, Nov 2018, pp. 1–5.

[7] W. Barth, *Nagios: System and Network Monitoring*, 2nd ed. San Francisco, CA, USA: No Starch Press, 2008.

[8] "Zabbix.org," https://zabbix.org/wiki/Main_Page, retrieved Feb, 2019.

[9] "NewRelic," https://newrelic.com/press-release/20150506-2, retrieved Feb, 2019.

[10] "Dynatrace," https://www.dynatrace.com/capabilities/microservices-and-container-monitoring/, retrieved Feb, 2019.

[11] F. Pina, J. Correia, R. Filipe, F. Araujo, and J. Cardroom, "Nonintrusive monitoring of microservice-based systems," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–8.

[12] "Amazon CloudWatch," retrieved Feb, 2019. [Online]. Available: https://aws.amazon.com/cloudwatch/

[13] "Azure Monitor," retrieved Feb, 2019. [Online]. Available: https://azure.microsoft.com/en-us/services/monitor/

[14] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 74, 2003.

[15] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang, "vpath: Precise discovery of request processing paths from black-box observations of thread and network activities," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, ser. USENIX'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 19–19.

[16] B. H. Sigelman, L. Andr, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper , a Large-Scale Distributed Systems Tracing Infrastructure," Tech. Rep. April, 2010.

[17] "Principled workflow-centric tracing of distributed systems," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 401–414.

[18] "Linkedin - tracing," https://engineering.linkedin.com/distributed-service-call-graph/real-time-distributed-tracing-website-\-performance-and-efficiency, retrieved Feb, 2019.

[19] "Apache Samza," retrieved Feb, 2019. [Online]. Available: http://samza.apache.org/

[20] "Netflix- tracing," https://speakerdeck.com/adriancole/distributed-tracing-and-zipkin-at-netflixoss-barcelona, retrieved Feb, 2019.

[21] "Opentracing," http://opentracing.io/, retrieved Feb, 2019.

[22] "Opencensus," https://opencensus.io/, retrieved Feb, 2019.

[23] S. Haselböck and R. Weinreich, "Decision guidance models for microservice monitoring," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, April 2017, pp. 54–61.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.