On black-box monitoring techniques for multi-component 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*—Despite the advantages of microservice and function-oriented architectures, there is an increase in complexity to monitor such highly dynamic systems. In this paper, we analyze two distinct methods to tackle the monitoring problem in a system with reduced instrumentation. Our goal is to understand the feasibility of such approach with one specific driver: simplicity. We aim to determine the extent to which it is possible to characterize the state of two generic tandem processes, using as little information as possible.**

To answer this question, we resorted to a simulation approach. Using a queue system, we simulated two services, that we could manipulate with distinct operation sets for each module. We used the total response time seen upstream of the system. Having this setup and metric, we applied two distinct methods to analyze the results. First, we used supervised machine learning algorithms to identify where the bottleneck is happening. Secondly, we used an exponential decomposition to identify the occupation in the two components in a more black-box fashion.

Results show that both methodologies have their advantages and limitations. The separation of the signal more accurately identifies occupation in low occupied resources, but when a service is totally dominating the overall time, it lacks precision. The machine learning has a more stable error, but needs the training set. This study suggest that a black-box occupation approach with both techniques is possible and very useful.

***Index Terms*—Black-box monitoring, Observability, Simulation, Analytics**

I. Introduction

Microservice systems are a modern approach used by major technological companies to create highly available, elastic and dynamic systems. This kind of architecture enables teams to work independently in different life-cycles and ensures that modules are oblivious to changes in the surrounding system. Despite the changes in the development methodology, there are some challenges concerning monitoring and observability of production systems, as these become more complex. Dynamic scalability, network distribution, third-party resources, or dynamic architectures, are the sort of features that make microservice systems so difficult to observe and control. When compared to monolithic solutions, this increases the possible points of failure and can severely decrease the quality-of-service. We need better tools and methods to understand the overall status of the system. More traditional approaches based on filling the source code with instrumentation, or generic metrics such as CPU or network occupation, agents, logging, watch-dogs, dashboards, etc., have serious shortcomings for modern distributed systems. Although they can indeed create a very good image of the infrastructure, two problems subsist: first, they only supply an extended set of tools to react to already in-place incidents, and secondly they lack "intelligence", if the system does not have observability, i.e., if we cannot access metrics in precise spots that are hidden inside complex black boxes.

Unfortunately, actual monitoring tools are built on the premise of observable systems, with agents, instrumentation, or some sort of module heartbeats. Furthermore, some resources may elude administrators control, such as objects located in third-party providers - e.g. content delivery networks. While some frameworks aim to gather information from the client-side point-of-view such as Pingdom [1], Bucky [2], or enterprise solutions, such as Dynatrace [3], they are basically aimed at creating simple dashboards and insights of the platform to trigger alerts to administrators based on a set of custom rules. We want to go beyond this, and automatically infer service occupation, using as little data as possible from the systems, possibly because such data is unavailable.

In [4], we gathered clients' data — collected by JavaScript snippets —, to improve monitoring using the client-side point-of-view, as a complement to traditional monitoring applications. In [5], we used machine learning techniques to pinpoint two sources of system bottlenecks — CPU and network —, using only the raw data visible by clients.

In this paper we further extend our previous work with two generic layers that simulate two components of a system, as in [5], where internal observability may not apply. This system could be a microservice that is complex and contains several queues inside, or a couple of microservices, one after the other, if intermediate timings of requests are not available, i.e., if we cannot relate the times at which the first and second services interact in response to an initial request to the first service. To get insights about the two-layer system, we only used the total time seen by the caller. This time aggregates the overall invocation time that sums up the two services. Based on these time, our goal is to determine the overall occupation of both services. We created a simulation using a two queue system, with one goal: understand the service occupation of each layer. We ran a simulation with two queuing systems and collected the response times. We then applied two methodologies to extract occupation of each layer (component) and extract error metrics: first, we used a similar approach as in [5], with supervised machine learning algorithms to identify service capability. Secondly, we used another method that tries to decompose the overall response time into the components' times to identify the occupation of each.

Our results demonstrate that a methodology that extrapolates information about a non-observable system of two layers is feasible and can improve performance monitoring. There is no overhead associated with these methodologies, and both methods — machine learning and division of the signal —, present advantages and complementary properties. These methods can improve monitoring when instrumentation or observability is difficult or unable to be achieved by administrators of the system.

The rest of the paper is organized as follows. Section II describes the monitoring problem we tackle in this paper as well as the methods we propose. Section III analyze how we made our experiment. In Section IV we present and evaluate the meaning of the results, discussing the strengths of both approaches and its limitations. In Section V we present the related work and finally in Section VI concludes the paper and describes future directions.

II. Proposed Methodology

In this section, we describe the problems and challenges associated with observability of a system and the definition of the metrics used.

With the increase of complexity of applications, and the need to reduce time-to-market, monitoring becomes more important than ever to ensure that component failures and bottlenecks do not affect user's quality-of-experience. Traditional monitoring approaches use a large set of tools and methods, such as tracing, logging, correlation identifiers between services or system tools, such as `Nagios` [6] or `Zabbix` [7]. Although functional, these approaches require the system to be prepared to give away information about its current internal status. In legacy systems, or systems without some form of instrumentation or agents, this might be difficult. Additionally, the effort to create logging or tracing in a production system may be too high.

In this paper, we present an approach that neither requires instrumentation, nor disperse logging tools over the system. In [5], we followed a similar approach to pinpoint bottlenecks in two distinct layers: external network and internal system. In this paper, we further evolve this methodology, by modeling a system in two components, and determining each component's occupation, without using instrumentation in the middle. Understanding each component's occupation may give a huge advantage for administrators, whenever monitoring is impracticable, either because it is too costly or because the internal details of the system are unknown, e.g., because the source code is unavailable or too complex. The only metric that we used was the total time observed by the entity that evoked the system. The total time represents the time that the request spent in the two components, from the beginning of the request until the end. In fact, our method of decomposing a system of two layers is very generic and can be used in different scenarios, like a system with a database and network, or two modules interconnected, or even a chain of microservices.

In the next subsections, we present our methods to evaluate a "black-box" approach of a two-layer system. First, we present an approach based on a machine learning supervised algorithm. Then, we describe our method based on the split of the signal. Both methods use the data collected from our experiment in Section III.

A. Machine Learning Algorithm

We followed a machine learning approach to predict each layer's occupation from the experiment input data. We used a regression model — instead of a classifier — since our output is a continuous value, instead of a set of class labels [8]. We created a regression model for each layer, as illustrated in Figure 1.

In addition to the regression models, we also wanted to evaluate the possibility of having decision tree classifiers. This is very appealing, since decision tree models are highly interpretable, and therefore an advantage to system administrators and operators.

We train our model with $3,000$ samples, or lines. Each line has $2,000$ requests made to the system, meaning that our classifier has $2,000$ features – e.g. inputs. There is a 2001^{st} value in each line, that corresponds to the layer 1 or layer 2 occupation (i.e., the real occupation of each layer). This value is mandatory, since we are training a supervised machine learning algorithm. However, this output is not available in the test cases, because we want to predict the occupation level, as illustrated in Figure 1.

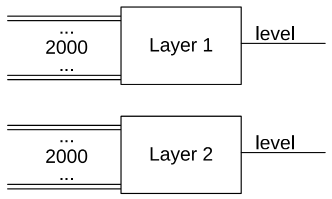

Fig. 1: Representation of the regression models.

Among the wide range of supervision machine learning methods available in the literature [9], we focused on two models: Simple Linear Regression (SLR) and Support Vector Machine (SVM). SLR assumes linearity between input and output, and SVM allows a non-linearity between input-output, to the kernel. This turns them into good choices to do the regression. Since we can manipulate the SVR kernel, we can use out-of-the-box a non-linear kernel, like a Radial Basis Function (RBF) or a polynomial kernel.

The literature [9] also includes a wide range of decision tree models, but since we wanted to predict a real value, we opted for a decision tree regression model. This kind of model allows us to have more insight about how it works, thus making it more interpretable, a great advantage for sysops.

To run the algorithms, under our training set, we used the `scikit` learn framework [10]. This framework, written in python, is a common standard for data scientists and people that want to generate models based on input data. For SVR, we used RBF and a polynomial kernel to evaluate the differences, with normalized data (input and output), in the $[0, 1]$ interval, to achieve better predictions. The decision tree regression model was evaluated with default parameters and data. Concerning the evaluation, all experiments were performed using a 10-fold cross validation with 20 repetitions.

B. Exponential Variable Sum Algorithm

The problem of determining the occupation of two sequential components, which we call layers, be they resources or distributed services, observing only global response times, can be modeled as determining the two tandem random processes responsible for generating an observed request time distribution. In other words, given a statistical distribution, since we know *a priori* that the distribution is the sum of two factors, what we want is to separate these factors. Particularly, in this paper we explore this approach under the assumptions that the service times are approximately exponentially distributed and the service is under relatively low load (occupation $\rho < 0.3$), at the time of observation. As long as the load and parallelism of each layer is known, the results can be used to extrapolate the total capacity of each layer. It should be noted however that this does not reduce the generality of the solution. The same approach could be applied under different distributions and loads, as long as some conditions are met (the resulting distribution having enough information to extrapolate the original processes). Furthermore, the two processes under study do not necessarily have to be tandem, they may, for example, be interleaved, as long as the sum of the components follows the previously described assumptions. A good example of where interleaving is the more adequate model, would be resource time-slicing.

To define it more precisely, given $f(x)$ and $g(y)$, which are the density functions of the service times of two layers, known to be approximately exponentially distributed, of rates λ and μ, we know that the total service time will be described by the sum of two random variables, $h(z)$, obtained from the convolution of the two, as shown in Equation 1.

$$h(z) = (f * g)(z) = \int_{-\infty}^{+\infty} f(z-y)g(y)\ dy \tag{1}$$

By substituting for the particular case of the exponential distribution, we get the result for $h(z)$ in Equation 2, which we integrate in Equation 3 to obtain the cumulative distribution function $H(z)$.

$$\begin{aligned} h(\lambda,\mu,z) &= \int_0^z \lambda e^{-\lambda t}\mu e^{-\mu t}\ dt \\ &= \lambda\mu\left(\frac{1}{e^{z\lambda}\mu - \lambda e^{z\lambda}} - \frac{e^{-z\mu}}{\mu-\lambda}\right) \end{aligned} \tag{2}$$

$$\begin{aligned} H(\lambda,\mu,z) &= \int_0^z h(t)\ dt \\ &= \int_0^z \lambda\mu\left(\frac{1}{e^{t\lambda}\mu - \lambda e^{t\lambda}} - \frac{e^{-t\mu}}{\mu-\lambda}\right)\ dt \\ &= \frac{\lambda\mu e^{-z\mu}\left(\lambda e^{z\lambda} + \left(\left(e^{z\lambda}-1\right)\mu - \lambda e^{z\lambda}\right)e^{z\mu}\right)}{\lambda e^{z\lambda}\mu^2 - \lambda^2 e^{z\lambda}\mu} \end{aligned} \tag{3}$$

Armed with the cumulative distribution function, and the empirical cumulative distribution function, $ecdf(x)$, from the observed sample S of total service time, we made an `R` script to determine the variables λ and μ, by solving the optimization model in Equation 4. The objective is selecting the variables to minimize the mean square error between the empirical cumulative function of the sample and $H(z)$, effectively fitting it to the data.

$$\begin{aligned} \underset{\widehat{\lambda},\widehat{\mu}}{\text{Minimize}} \quad & \frac{1}{n}\sum_{i=1}^{n}(\eta_i - \epsilon_i)^2 \\ \text{subject to} \quad & \tau = max(S_i),\ \forall i \\ & \delta = \frac{\tau}{1000} \\ & \eta_i = H(\widehat{\lambda},\widehat{\mu},i\delta),\ i = 1,\dots,1000 \\ & \epsilon_i = ecdf(i\delta),\ i = 1,\dots,1000 \\ & \widehat{\lambda} > 0,\ \widehat{\lambda} \in \mathbb{R} \\ & \widehat{\mu} > 0,\ \widehat{\mu} \in \mathbb{R} \end{aligned} \tag{4}$$

At the end of this step, we are left with a $\widehat{\lambda}$ and $\widehat{\mu}$, which are estimators of the service rate of Layers 1 and 2 respectively. Given that the load under which the measurements were taken, which we denote as W, as well as the parallelism levels, denoted c_1 and c_2, the occupation of each layer, ρ_1 and ρ_2 can be determined as shown in Equation 5 where S denotes the service rate of a layer ($\widehat{\lambda}$ or $\widehat{\mu}$).

$$\rho = \frac{W}{c \cdot S} \tag{5}$$

The assumption that the parallelism level c is known might seem limiting, however, in practice this parameter is easy to monitor using classical tools, and even if not directly measurable, heuristics can be used to estimate it. One such example is exploiting the relationships between maximum throughput T and parallelism given by Equation 6.

$$T = c \cdot S \tag{6}$$

III. Evaluation

We validated and benchmarked the two methods with total service time samples extracted from a simulated service with two layers. The described system was simulated as two sequential single server queue systems ($M/M/1$) in `R` using the `qcomputer` [11] package (shown in Figure 2).

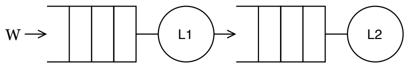

Fig. 2: Two sequential single server queue systems.

To simulate different occupation rates, we used a fixed global request arrival rate W of 30 requests per unit of time and varied the service rates S of each layer according to Equation 5. A set of 30 samples of 2000 observations each was then obtained for all the permutations of the two layers with occupations in the range $[0.1, 0.9]$ in 0.1 increments (e.g. $(0.1, 0.1), (0.1, 0.2), ...$). Each sample, a line in a `CSV`, contains 2000 features and the 2 targets, in this case the occupation parameters.

Using cross-validation we trained the machine learning algorithms and evaluate the quality of the predictions. For the exponential variable sum algorithm we used the proposed optimization approach to predict the targets for each sample.

To compare the two general approaches we calculated the Mean Square Error (MSE) for each layer grouped by occupation range in 0.1 increments. Additionally, as we tried multiple machine learning approaches, these metrics were also used to pick the best algorithm.

IV. Results

As we referred in Section II, we collected a set of 2000 total response time for the clients invocations. We analyze the results obtained with the machine learning algorithms and also with the exponential decomposition.

TABLE I: Regression model results for Layer 1 and Layer 2 occupation

	Layer 1	Layer 2
Method	MSE	MSE
Decision Tree	0.12 ± 0.08	0.09 ± 0.06
SLR	0.81 ± 0.61	0.93 ± 0.44
SVR	0.05 ± 0.04	0.05 ± 0.03

The results obtained for the Layer 1 and 2 occupation models are summarized in Table I, Table II and Table III. We report average results for the mean square error (MSE), between the predicted and actual values. Since we performed 20 repetitions of 10-fold cross-validation, we present average and standard deviation results.

Table I presents the results for three distinct classifiers: decision tree regressor, SLR and SVR algorithms. As expected, SVR outperforms the others algorithms, since it uses a non-linear kernel that fits better the raw data that is handled in this use case scenario.

TABLE II: Machine Learning Decomposition

Range	Layer 1 - MSE	Layer 2 - MSE
0.1	0.12	0.12
0.2	0.07	0.08
0.3	0.04	0.03
0.4	0.03	0.02
0.5	0.02	0.02
0.6	0.03	0.02
0.7	0.03	0.03
0.8	0.03	0.03
0.9	0.03	0.03

TABLE III: Exponential Decomposition

Range	Layer 1 - MSE	Layer 2 - MSE
0.1	0.08	0.08
0.2	0.12	0.26
0.3	0.25	0.51
0.4	0.40	0.68
0.5	0.93	0.74
0.6	1.79	1.76
0.7	3.76	4.95
0.8	11.38	11.43
0.9	66.48	74.87

The SVR algorithm attained an average of 0.05 MSE (with 0.04 standard deviation) for Layer 1 and 0.05 (with 0.03 standard deviation) for Layer 2. On a $[0, 1]$ range this is a good result.

SVR algorithm outperforms SLR and Decision tree model, for both layers. Therefore, we compared SVR algorithm with a non-linear kernel with the exponential decomposition method. To do this, we trained the same SVR algorithm for both Layer 1 and Layer 2. We submitted to the fitted model only values for a specific range (e.g. 0.1, 0.2) and registered the MSE for the predicted values. In Table II we can evaluate the machine learning accuracy for distinct Layer levels. We can observe that for low occupied layers the algorithm have a worst MSE, compared with a more occupied Layer.

In Table III we can observe the algorithm to split the two layers with the exponential decomposition method. We evaluated the MSE for each Layer 1 and Layer 2 occupation level. One could notice that this algorithm outperforms SVR on lower occupations levels. However, for high occupation levels, the exponential algorithm, does not predict correctly, being outperformed by the machine learning approach.

Another difference is that the Machine Learning algorithm requires supervised training and the exponential decomposition does not. On the other hand, the latter can only be used when the system load during the sampling period is known. Given their complementary properties and ranges of effectiveness, they are obvious candidates for hybrid application.

V. Related Work

The monitoring research field is active with several contributions in the academy as well as in the industry. Hence, we present several works from both fields.

In the industry, companies such as New Relic [12], DynaTrace [3] or distributed tracing system such as ZipKin [13], aim to give frameworks or tools to administrators, to create dashboards or notifications. Our work focus on increasing trustworthiness and reliability of the overall system with autonomous prediction of system occupation.

In academic research, [14] uses an approach with a architecture where each microservice makes self-management concerning monitoring and scaling. This approach may lead to the discard of the importance of call paths, and the "waterfall" effect that a microservice may have in others components. In [15], a tool based in the global entropy of a distributed system is presented to automatically detect anomalies. However, due to the fact that do not rely on response time and other performance metrics it may lead to false positives. In [16], [17], Malkowski *et al.* studied bottlenecks in N-tier systems, to analyze multi-bottlenecks, due to saturation. They managed to conclude that lightly loaded resources may be responsible for that phenomenon. In [18], authors try to discover bottlenecks in data flow programs running in the cloud. They focus more on CPU and I/O bottlenecks, and not predicting occupation.

In [19] the goal is to model web servers as single server queues in terms of response time and overall system performance. [20] uses a similar approach for an Apache Web Server. In [21], layered queueing networks are used to model a system with two layers – frontend and backend. [22] presents a model for Multi-tiered Web Applications using queues for individual components such as CPU, I/O or Network. However, there are multiple points of observability, being this way a solution that although functional is more intrusive than ours. Our premise is simplicity. We aim to automatically determine the overall occupation of the system layers from exterior observations, meaning the method requires no additional instrumentation nor complex measurements.

VI. Conclusions and Future Work

Monitoring and observability of systems is a challenge for administrators, due to increased application elasticity, complexity, granularity and dynamics. The current tools put the burden of analysis, such as detecting bottlenecks and performance issues on the operators. As the number of services and technologies in a system increases, so does the complexity and time spent in operational tasks, such as root cause analysis.

Our goal is to identify the occupation of each component on a two layer system. The evidence collected shows that it is possible to identify the capacity and occupation of each layer, solely from the overall response time, as measured by a client. Since the proposed method is not coupled to any infrastructure or language, it can be used in any kind of system where observability is a major concern. The two methods – supervised machine learning and exponential decomposition – combined can complement each other's shortcoming and provide insight about overall system performance.

As future work, there are several directions we want to explore. Since the total request time distribution carries enough information to infer the capacity of each system component, we want to generalize our model, to more than a two layer system. Secondly, we want to test our method in more setups to ensure its robustness. Finally, since our ultimate goal is improving monitoring techniques and providing better insights to administrators, we would like to implement this method in a real world scenario.

Acknowledgment

This work was partially 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] "Pingdom," https://www.pingdom.com/, retrieved: Jun, 2017.
[2] "Bucky," http://github.hubspot.com/bucky/, retrieved: Jun, 2017.
[3] "Dynatrace," https://www.dynatrace.com/platform/, retrieved May, 2017.
[4] R. Filipe and F. Araujo, "Client-side monitoring techniques for web sites," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Oct 2016, pp. 363–366.
[5] R. Filipe, R. P. Paiva, 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, pp. 1–5.
[6] "Nagios," https://www.nagios.org/, retrieved September, 2018.
[7] "Zabbix," https://www.zabbix.com/, retrieved September, 2018.
[8] A. Sen and M. Srivastava, *Regression analysis: theory, methods, and applications*. Springer Science & Business Media, 2012.
[9] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Burlington, MA: Morgan Kaufmann, 2016.
[10] "Scikit learn," http://scikit-learn.org, retrieved Jun, 2018.
[11] A. Ebert, P. Wu, K. Mengersen, and F. Ruggeri, "Computationally Efficient Simulation of Queues: The R Package queuecomputer," mar 2017.
[12] "New Relic," https://newrelic.com, retrieved May, 2017.
[13] "Zipkin," http://zipkin.io/, retrieved Oct, 2017.
[14] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds, "An architecture for self-managing microservices," in *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, ser. AIMC '15. New York, NY, USA: ACM, 2015, pp. 19–24.
[15] H. Malik and E. M. Shakshuki, "Towards identifying performance anomalies," *Procedia Computer Science*, vol. 83, no. Supplement C, pp. 621 – 627, 2016, the 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.
[16] S. Malkowski, M. Hedwig, J. Parekh, C. Pu, and A. Sahai, "Bottleneck detection using statistical intervention analysis," in *Managing Virtualization of Networks and Services*. Springer, 2007, pp. 122–134.
[17] S. Malkowski, M. Hedwig, and C. Pu, "Experimental Evaluation of N-tier Systems: Observation and Analysis of Multi-Bottlenecks," *IISWC '09: Proceedings of the 2009 IEEE 12th International Symposium on Workload Characterization*, 2009.
[18] "Detecting bottlenecks in parallel DAG-based data flow programs," *2010 3rd Workshop on Many-Task Computing on Grids and Supercomputers*, pp. 1–10, 2010.
[19] 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.
[20] T. Van Do, U. R. Krieger, and R. Chakka, "Performance modeling of an apache web server with a dynamic pool of service processes," *Telecommunication Systems*, vol. 39, no. 2, pp. 117–129, 2008.
[21] Y. Shoaib and O. Das, "Web application performance modeling using layered queueing networks," *Electron. Notes Theor. Comput. Sci.*, vol. 275, pp. 123–142, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2011.09.009
[22] A. Kattepur and M. Nambiar, "Performance modeling of multi-tiered web applications with varying service demands," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015, pp. 415–424.