International Journal Of Mechanical Engineering And Information Technology
An Official Publication Of IGM Publication

Detection of Performance Anomalies in Cloud Services: A Correlation Analysis Approach

Authors
Ghodai M. Abdelrahman[1], Mona M. Nasr[2]
Department of Information Systems Faculty of Computers & Information, Helwan University Ain Helwan, Helwan, Cairo, Egypt
Email- eng.ghady_mohammad@hotmail.com, drmona_nasr@fci.helwan.edu.eg

ABSTRACT
The prevalence of ubiquitous computing and communication has coined the term of cloud computing through which, software, infrastructure and platform can be provided as a service. Software as a service (SaaS) is getting an increasing potential as a cloud-based option for using software applications in a pay-per-use manner. A critical challenge in SaaS model is continuous attestation of the compliance with quality of service (QoS) metrics stated in SLAs. In this paper, we propose a method for detecting performance anomalies in cloud software services. The proposed method uses correlation analysis between computing resources utilization and workload characteristics. This is done by comparing the correlation values to a reference load test values performed before the SaaS deployment to identify deviations and notify the system administrator about it. The testing scenario operates in two steps. First, running a standard benchmark on a virtual machine to simulate workload and record the correlation between workload and available computing resources utilization (i.e., CPU, RAM, HDD, and Network). Second, the same benchmark is executed again but with changing the workload characteristics through injecting additional queries or changing the computing resources configuration values of the virtual machine. The changes are only present on specific time points to testify the detection rate. Results on standard benchmarks TPC-C, TPC-D and TPC-W showed a promising detection rate that can assure SLA targeted quality aspects such as reliability, scalability and security.

Keywords- *Software as a Service, Cloud Services, Performance Testing, Anomaly Detection, Cloud Computing, Correlation Analysis*

1. INTRODUCTION

Cloud computing is revolutionizing the software development industry as a promising deployment paradigm. It provides computing resources as services in a pay-per-use usability pattern. This facilitates saving a plenty of time and cost of maintaining hardware and software components of applications. Accordingly, the majority of software development firms migrates its software products to the cloud. This migration poses new challenges, as the software testing procedures should deal with dynamic characteristics of the cloud. One of these challenges is the dynamic changes in the cloud resources over time depending on supply and demand. This necessitates a continuous monitoring of the provided service's performance anomalies to ensure early detection and handling of bottlenecks to avoid expensive loss.

Software performance anomaly is defined as a deviation in one of the two most popular metrics: *response rime* (or *latency*) and *throughput* due to changes in workload or computing resources [1]. Performance anomalies are barriers to achieving predictable performance guarantees in enterprise applications and often come with significant cost implications. As an example, Amazon.com website turned down for roughly 45 min in August 2013 due to an unexpected fault, which resulted in an estimated loss of 5 million dollars. Identifying performance anomalies can save up to 75% of failure recovery time, and doing it in a timely manner could inhibit 65% of failures, according to

Oppenheimer et al. [2]. These implications show not only the importance but also the potential economic value of robust and automated solutions for detecting performance problems in real time.

Fluctuations in cloud's supply and demand result in dynamic changes of resource allocation and workload which increases the possibility of software performance anomalies. This is in addition to the conventional causes found in web applications such as deployment errors or hardware failures. As a result, the monitoring of software services on the cloud is far complicated and time-consuming to be manually handled by systems administrators. Given a continuous automated monitoring procedure, the system administrator can have early alarms on possible failures before they trigger unforeseen service downtime. Therefore, it can save expensive economic loss to the service provider and his tenants.

The focus of this paper is to propose a continuous performance anomaly detection method that can deal with the dynamicity of cloud computing environments. The main contributions of the proposed method can be summarized as follows:

1. Monitoring performance metrics for each service tenant based on QoS aspects defined by SLA.
2. Detecting performance anomalies based on workload clusters to reduce the number of false alarm rates.
3. Online monitoring of SaaS performance in a continuous manner.

The first contribution ensures that the monitoring process would be aware of the differences in target performance metrics for each tenant of the cloud service, which is a vital point as different tenants will have different SLAs based on their requirements and rent amount. The second one guarantees the minimum false alarm rates which may result from dealing with workload as a single profile. Therefore, the workload is clustered based on quartile ranges to create reference profiles used during judgments on performance anomalies. Finally, the third one emphasizes the continuity of the monitoring process during the operation of the cloud service.

The rest of this paper is organized as follows. Section 2 explores the relevant literature. Section 3 presents the proposed methodology. Section 4 illustrates the experimental design and results. Section 5 concludes the paper and indicated the possible future work.

2. RELATED WORK

In this section, we are going to explore the relevant literature for workload modelling of web applications in general and cloud applications in specific. We explore literature in two basic parts. The first part presents workload modelling in web and cloud applications. The second one shows performance anomaly detection methods.

2.1 Workload Characterization and Modelling

A bottleneck is a resource or an application component that limits the performance of a system [1]. Malkowski et al.[3] describe a bottleneck component as a potential root-cause of undesirable performance behavior caused by a limitation (e.g., saturation) of some major system resources associated with the component[4]. Such components often exhibit frequent congestion of load [5]. Also, application or system metrics correlating with an observed performance limitation are referred to as bottleneck metrics [6].

Types of bottlenecks are defined according to [1] as two types, resource saturation bottlenecks and resource contention bottlenecks. In resource saturation bottlenecks, a resource is saturated when its capacity is fully utilized or past a set threshold. Moreover, saturation may also be estimated in terms of the length of a resource queue of jobs or request to be served by that resource. Saturation causes different system resources to be bottlenecked differently with varying performance impact.

While in resource contention bottlenecks, application processes contend for limited system resources such as CPU cycles, IO bandwidth, and physical memory, and also software resources such as buffers, queues, semaphores, and mutexes. The impact of such contention is well pronounced in

cloud data centers due to resource interference between multiple cloud tenants. The *noisy neighbours* effect is an analogy for this interference [7].

Characterization of load patterns on Content Distribution Networks (CDN) has been introduced by Jung et al. [8]. They proposed new ways of characterizing flash events, such as rejecting the service request of the users under a DOS attack. They proposed adaptive algorithms based on caching and dynamic delegation. Such algorithms are relevant to cloud load prediction in identifying load data generated by genuine (legitimate) users and not attackers.

A holistic workload analysis of web applications was explored by Lacort et al. [9]. The authors collected the workload data over long time period (weeks) with a huge users load simulation (greater than 50,000 users and roughly122 million requests). The stated that graphic files are the most common files in the Web, comprising more than 60% of the total web requests. It has also been reported that there was a noticeable increase of requests for dynamic pages against a decrease in the demand of static pages, the thing that showed that new generation of dynamic Web applications is gaining popularity. This study serves as a good model for similar characterization studies needed for cloud workloads.

Prediction of traffic congestion in web servers has been investigated by Baryshnikov et al. [10]. Authors monitored page request traffic in bus and airline transportation networks with focus on hotspot detection. A hotspot can be defined as an abrupt peak in traffic amount. This is relevant to the elasticity constraint in the cloud for satisfying the service level agreement (SLA) demands during hotspots. The authors proposed an autoregressive regression model [11] for hotspot detection using user's session request data. Their results showed feasible accuracy to predict seasonal hotspots such as Christmas holidays. While their study was basically on web applications, their objective of predicting hotspots in traffic is still needed in cloud services.

Arlitt and Williamson et al. [12,13] aimed to characterize web applications workload through analysing transferred documents' type and size distributions. They indicated that transferred documents can reflect the behaviour and geographical distributions of a web server's requests. Authors used six different web request data sets. The study showed that the mean size of the transferred documents was between 5 and 21 KB. Almost 90% of the transferred documents was HTML and image files. Techniques such as caching showed the ability to enhance the web server response time.

Methods for the management of virtualized environments in the cloud were introduced by Andreolini et al. [14]. The authors focused on guidelines for supporting virtual machine (VM) migration decisions in a cloud environment. Study proposed management algorithms for deciding about reallocations of virtual machines in a cloud context characterized by large numbers of hosts. Their results showed that proposed algorithms can be robust to variable contexts with ability to limit VMs migrations to only necessary situations.

2.2 Performance Anomaly Detection Methods

Tan et al. [15] presented a predictive performance anomaly prevention system (PREPARE) for virtualized cloud systems. PREPARE uses a Hidden Markov Model (HMM) to predict the values of attributes then, it selects suspicious attributes with tree augmented naïve Bayes (TAN) model [16].

Wang et al. [17] presented a workload-aware online anomaly detection method for enterprise applications. Authors utilized an incremental k-mean method to differentiate workloads then, they used local outlier factor (LOF) [18] to detect anomalies in each specific workload pattern. However, this method has two basic limitations. First, it is unable to automatically determine the number of clusters in the data as it is given as an input parameter. Second, the method computes LOF for every arrival data instance, and the computation complexity increases significantly as the number of data instances increases.

Mi at al. and Zhang at al. [19,20] provided performance anomaly detection methods that are based on domain specific knowledge (e.g., the system internal structure). It can pinpoint to specific application's components which contribute to performance bottlenecks. Although this is useful to locate faulty parts in the application, it is dependent on specific application design, which makes it difficult to generalize on other designs.

3. METHODOLOGY

This section introduces the mathematical basis of the proposed method, its algorithmic formulation and the supposed operation workflow.

The proposed method depends on finding the correlation between the workload represented by user load and computing resources utilization. To achieve this goal, we used the Multivariate Normal (MVN) distribution [21] to estimate the probability distribution of the computing resources with regard to each user load range. While real data are not exactly multivariate normal, the normal density is often a realistic approximation to the "true" population distribution according to *the central limit theorem* [22].

MVN extends the univariate version by having $\boldsymbol{\mu}$ to be a vector instead of a scalar value such $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_n)$ and $\Sigma = cov(\boldsymbol{x})$ is the variance-covariance matrix $(n \times n)$ whose diagonal terms are variances and off-diagonal terms are covariances. It can be calculated by the following equation:

$$\Sigma = E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] \quad (1)$$

Definition 1

The random vector $\boldsymbol{x} \forall \boldsymbol{x} \in \mathbb{R}^n$ is said to be multivariate normal if and only if the linear function

$$\boldsymbol{a} \cdot \boldsymbol{x} = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

is normal for all $\boldsymbol{a} = (a_1, a_2, \cdots, a_n)$.

Theorem 1

If $\boldsymbol{x}$ is a n-variate normal with mean $\boldsymbol{\mu}$ and covariance matrix Σ, then $\boldsymbol{x}$ has a probability density function (pdf) given by

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \quad (2)$$

Based on the last mathematical basis, the proposed method utilizes the following algorithm to detect performance anomalies.

Algorithm 1

Input: Reference resource utilization data set $\{x^1, x^2, \cdots, x^m\} \forall x \in \mathbb{R}^n$ and a test vector x^i.

Processing:

1. Using the reference dataset, fit the MVN parameters $\boldsymbol{\mu}$ *and* Σ.
2. Calculate $p(x^i)$ using equation (2).
3. Anomaly if $\left|E(\boldsymbol{x}) - p(x^i)\right| > \varepsilon$.

Output: Anomaly alarm if x^i satisfies the anomaly condition.

In this study, each input vector represents the utilization percentage of four computing resource: CPU, memory, hard drive and network respectively. Each resource represents a feature for the proposed detection method. The anomaly condition is checking that the difference between the expectation of the MVN pdf and the probability of the new sample is greater than a threshold ε. The threshold value is set to the double of the standard deviation of the MVN pdf. This value showed to achieve the lowest false alarm rate (FAR) with feasible adaptability.

Fig. 1 shows a sample from MVN pdf of CPU and memory as it is the max feasible number of features to visualize in a 3D graph, but in our analysis we use the four features as input.

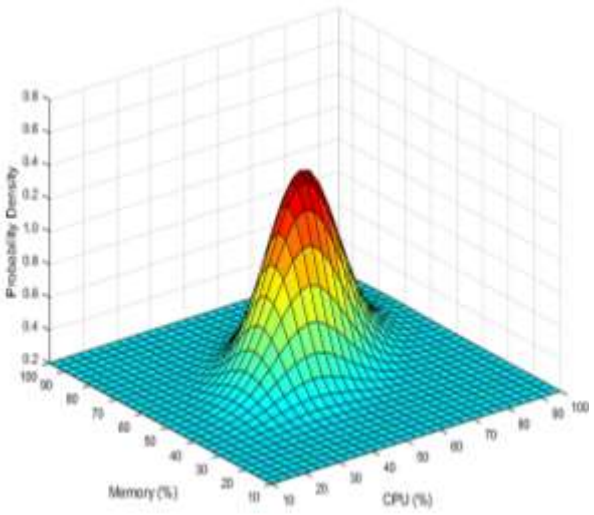

Fig. 1 Sample MVN pdf values for CPU and memory utilization features

We transformed the four features to precisely fit the Normal distribution. We calculate the $\log(x)$ for the CPU and Network features and $x^{1/3}$ for the memory and HDD features.

For an illustration of the operation workflow of the proposed method, Fig. 2 presents a diagram that describes the involved entities and operations during the anomaly detection process.

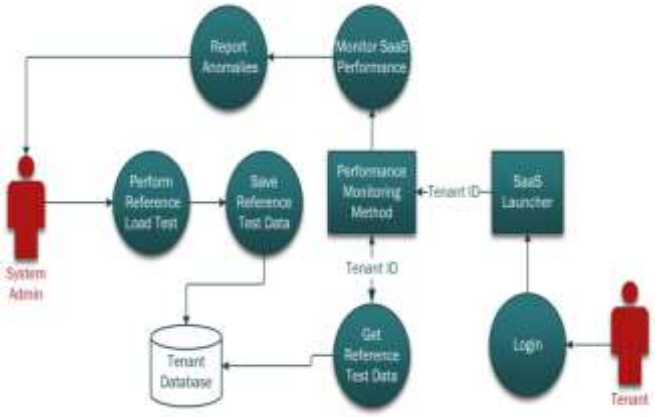

Fig. 2 Operation workflow of the proposed method

The workflow design comprises the following execution sequence:

a. The system administrator starts the workflow by performing a load test for the SaaS application according to a tenant's SLA that defines targeted performance metrics. Basically, this step is executed on the operation environment or at least an identical simulation environment in terms of computing resources and workload.
b. After performing the reference load test, the admin saves the test data in the tenant database using the tenant_ID.
c. When one of the tenant's users tries to launch the SaaS service instance using the tenant_ID, the SaaS launcher middleware will initiate new monitoring session passing the tenant_ID to the proposed monitoring process.
d. Given the tenant_ID the proposed monitoring process retrieves the reference load test data and uses it to fit the MVN model parameters.
e. At every configured time window length, the monitoring process calculates the average user load and resources utilization features vector to testify it against the configured anomaly threshold.
f. If the pdf value of the current time window satisfies the anomaly condition in Algorithm 1, the monitoring process sends an anomaly alarm to the system administrator to take a corrective action.

The proposed operation workflow is designed to fit the cloud computing operation style which is tenant-based working environment focused on defined SLAs metrics. Building our proposed method on the same concepts makes it able to:

- Manage multiple configured tenants' accounts.
- Help the SaaS providers to satisfy each SLA requirement.
- Reduce false alarm rates as the monitoring is tenant-based.

4. EXPERIMENTAL DESIGN AND RESULTS

This section shows the experimental design for testifying the proposed method and summarized the results.

As stated earlier, we are presenting the computing resources in the working environment by four basic features as shown in Table 1.

Table 1 Computing resource features

Name	Description
CPU	Average CPU utilization in percentage
Memory	Average RAM utilization in percentage
HDD	Average hard disk utilization in percentage
Network	Average network utilization in percentage

During the reference load test, we divide the user load values into four quartile ranges (each 25%). These quartiles are stored with its relevant resource utilization values. This helps to reduce the FARs by comparing each test example with its nearest quartile value.

Standard benchmarks including TPC-C, TPC-D and TPC-W [23] utilized to simulate different user workloads. These benchmarks are described in the following points.

- TPC-C is an on-line transaction processing (OLTP) benchmark. TPC-C involves a mix of five concurrent transactions of different

types and complexity either executed on-line or queued for deferred execution.

- TPC-D represents a broad range of decision support (DS) applications that require complex; long running queries against large complex data structures.
- TPC-W is a transactional web benchmark. The workload exercises a breadth of system components associated with such environments.

For each benchmark, we run two stages simulating the real proposed working scenario:

1. We only run the benchmark on the simulation environment (which is a virtual machine) and record the user load and its relevant resource utilization features.
2. We run the same benchmark, but we alter the virtual machine resource values or inject additional queries from another benchmark at random time points.

We used Dell's benchmarking software called "Benchmark Factory" to run the three configured benchmarks [24]. This useful tool provides a wide range of standard benchmarks in addition to its ability to configure, run and store multiple benchmarks simultaneously.

A description of utilized software in this experimental design is listed in Table 2 which shows all the utilized software and their role.

Table 2 Utilized software in the experimental design and its role

Software	Role
Benchmark Factory	Configuring benchmarks, executing it and storing load data.
ORACLE VM VirtualBox	For configuring and running virtual machines.
Microsoft SQL Server 2012	Stores the reference tests data per each tenant_ID. Moreover it is used to store anomaly alarm notifications.
.NET Framework (C#)	Used to implement the monitoring service as a windows service.
Matlab 2015b	For results visualization.

We configured each benchmark to run a maximum of 100 user load scaled gradually over 1 hour of runtime.

Running our experiments on virtual machines simulates the real working environment on the cloud while providing more control on the available computing recourses for each run.

For the hardware description, Table 3 states the hardware specifications of the utilized machine.

Table 3 Hardware specifications of the experimental design's machine

Hardware	Specification
CPU	Intel Core i7 (2.8 GHz, 8MB cache)
RAM	16 GB
Hard Disk Drive	1 TB
Network Bandwidth	35 Mbps

For sample results visualization, Fig. 3 shows the pdf values for the TPC-C benchmark in the first quartile range of user load. In Fig. 4, we show the same pdf values but with injecting additional queries by running another benchmark simultaneously. This is done at configured time points to testify the anomaly detection method.

As per our benchmark configuration, the time to run each quartile range of user load is roughly 15 minutes which is visualized on the y-axis. The x-axis shows the pdf values in terms of mean and standard deviation magnitudes.

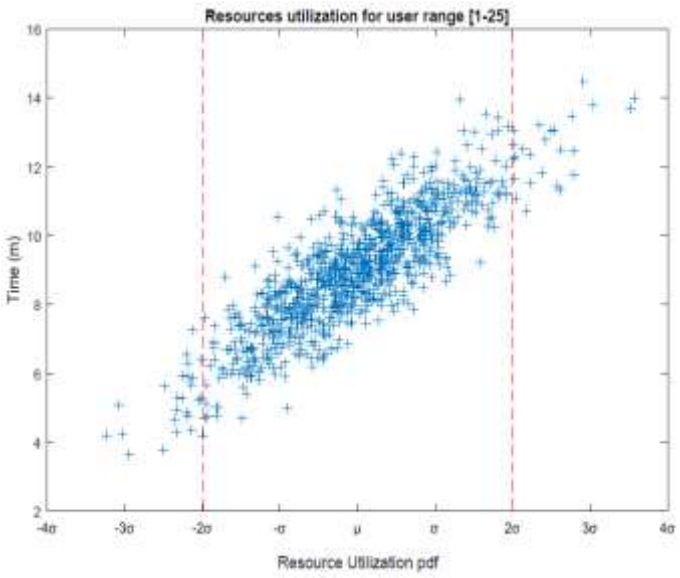

Fig. 3 MVN pdf values for the four resource utilization features (CPU, Memory, HDD and Network) of the first quartile range of user load [1-25] by running TPC-C benchmark.

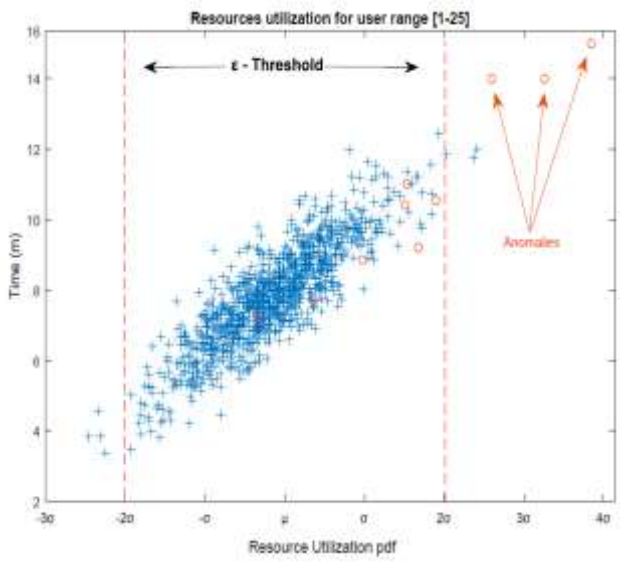

Fig. 4 MVN pdf values of the first quartile range of user load [1-25] by changing the workload of TPC-C benchmark.

The red dashed lines mark the anomaly threshold ranges based on the standard deviation. The outlier MVN pdf values have been marked by orange circles. Basically, these points represent the time points at which the anomaly represented by changes in computing resources or queries load.

For summarization of test results over all the three benchmarks, Table 4 lists the detection accuracy in terms of the percentage of truly detected anomalies.

Table 4 Detection accuracy in percentage of anomalies per each benchmark

Benchmark	Detection Rate (%)
TPC-C	98
TPC-D	96
TPC-W	94
AVG	96
STD	2

5. CONCLUSION

In this paper, an automatic performance anomaly detection method has been proposed for cloud services. The design of the proposed method depends on finding the correlation between computing resources and user load during a reference load test that is configured to fulfil a specific SLA of a tenant account. During the service operation, the monitoring procedure compares the average user load and its relevant average resource utilization against the reference test data of the closest quartile range of user load. The results showed an average of 96% of detection accuracy over the three configures benchmarks.

The main contribution of the proposed method can be summarized in three main parts. First, it stores and analyzes load data based on each SLA requirements as it is the same configuration mechanism in cloud services. This can help the service provider to customize the target performance objectives per each tenant and reduce false alarm rates. Moreover, it also can be considered as guarantee of other important non-functional requirements such as availability and reliability. Second, it compares anomalies with relevant user load using quartile ranges to provide better resolution and reduce false alarm rates. Finally, it provides the potential to early detect denial of service security attacks as it can be considered as a sudden change in the generated queries which has been simulated during our experimental design.

Our future work will be exploring the feasibility of other anomaly detection methods such as Local Outlier Factor LOF and Local Outlier Probability (LoOP) and investigating adaptive techniques for setting the anomaly detection threshold.

REFERENCES

1. B. Gregg, *Systems Performance: Enterprise and the Cloud*. Pearson Education, 2013.
2. D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *USENIX symposium on internet technologies and systems*, vol. 67. Seattle, WA, 2003.
3. S. Malkowski, M. Hedwig, and C. Pu, "Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 118–127.
4. D. Lee, S. K. Cha, and A. H. Lee, "A performance anomaly detection and analysis framework for dbms development," *IEEE Transactions on Knowledge and Data*

Engineering, vol. 24, no. 8, pp. 1345–1360, 2012.

5. N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Burstiness in multi-tier applications: Symptoms, causes, and new models," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc., 2008, pp. 265–286.
6. J. Parekh, G. Jung, G. Swint, C. Pu, and A. Sahai, "Issues in bottleneck detection in multi-tier enterprise applications," in *200614th IEEE International Workshop on Quality of Service*. IEEE, 2006, pp. 302–303.
7. X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010, pp. 51–58.
8. J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 293–304. [Online]. Available: http://-doi.acm.org/10.1145/511446.511485
9. J. A. Lacort, A. Pont, J. A. Gil, and J. Sahuquillo, "A comprehensive web workload characterization," in *Proc. of the Conference on Performance Modelling and Evaluation of Heterogeneous Networks*. Citeseer, 2004.
10. Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana, "Predictability of web-server traffic congestion," in *10th International Workshop on Web Content Caching and Distribution (WCW'05)*, Sept 2005, pp. 97–103.
11. J. D. Hamilton, *Time series analysis*. Princeton university press Princeton, 1994, vol. 2.
12. M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 631–645, Oct. 1997. [Online]. Available: http://-dx.doi.org/10.1109/90.649565
13. A. Williams, M. Arlitt, C. Williamson, and K. Barker, "Web workload characterization: Ten years later," in *Web content delivery*. Springer, 2005, pp. 3–21.
14. M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, *Dynamic Load Management of Virtual Machines in Cloud Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 201–214.
15. Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, June 2012, pp. 285–294.
16. F. Zheng and G. I. Webb, "Tree augmented naive bayes," in *Encyclopedia of Machine Learning*. Springer, 2011, pp. 990–991.
17. T. Wang, W. Zhang, J. Wei, and H. Zhong, "Workload-aware online anomaly detection in enterprise applications with local outlier factor," in *2012 IEEE 36th Annual Computer Software and Applications Conference*, July 2012, pp. 25–34.
18. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, May 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335388
19. H. Mi, H. Wang, Y. Zhou, M. R. T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed*

Systems, vol. 24, no. 6, pp. 1245–1255, June 2013.

20. Y. Zhang, Z. Zheng, and M. R. Lyu, "An online performance prediction framework for service-oriented systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 9, pp. 1169–1181, Sept 2014.
21. C. B. Do, "The multivariate gaussian distribution," *Section Notes, Lecture on Machine Learning, CS*, vol. 229, 2008.
22. A. Araujo and E. Giné, *The central limit theorem for real and Banach valued random variables*. Wiley New York, 1980, vol. 431.
23. M. Poess and C. Floyd, "New tpc benchmarks for decision support and web commerce," *ACM Sigmod Record*, vol. 29, no. 4, pp. 64–71, 2000.
24. "Benchmark factory," https://-software.dell.com/products/benchmark-factory/, accessed: 2016-09-05.