IntelligentAutomationAndSoftComputing,2020
Copyright © 2020, TSI® Press
Vol.26,no.2,353–365
https://doi.org/10.31209/2019.100000159

Dynamic Horizontal and Vertical Scaling for Multi-tier Web Applications

Abid Nisar[1], Waheed Iqbal[1], Fawaz Bokhari[1], Faisal Bukhari[1], Khaled Almustafa[2]

[1]Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan
[2]Prince Sultan University, Riyadh, KSA

ABSTRACT
The adaptive resource provisioning of cloud-hosted applications is enabled to provide a better quality of services to the users of applications. Most of the cloud-hosted applications follow the multi-tier architecture model. However, it is challenging to adaptively provision the resources of multi-tier applications. In this paper, we propose an auto-scaling method to dynamically scale resources for multi-tier web applications. The proposed method exploits the horizontal scaling at the web server tier and vertical scaling at the database tier dynamically to maintain response time guarantees. We evaluated our proposed method on Amazon Web Services using a real web application. The extensive experimental results show the effectiveness of our proposed method in terms of performance and cost when compared with current practices of static and dynamic resources over-provisioning methods.

KEY WORDS: Cloud computing, auto-scaling, multi-tier, web applications, vertical database scaling, horizontal web scaling.

1 INTRODUCTION

CLOUD computing has emerged as a promising technology for the provision of low cost, on-demand, and pay-as-you-go services to enterprises, application service providers, and individual users. Web applications are widely hosted on the cloud to avail these benefits. Nowadays, most of the large scale web applications follow multi-tier architecture. In a typical multi-tier architecture, a web server and database server are deployed independently and they both interact with each other to maximize the performance and scalability. Usually, a multi-tier web application is hosted on the cloud using an Infrastructure-as-a-Service (IaaS) model, which allows a significant control on the resources to control and manage. A traditional approach for deploying a multi-tier web application on IaaS is to provide and use Virtual Machines (VMs) for web and database server tiers separately.

Cloud computing offers availability and reliability level and service level agreements Shah, S. C. (2017); Abdullah et al. (2018). However, there are limited guarantees offered by the cloud providers for the performance of the applications hosted on the cloud. Mostly, the owners of the applications have to manage the performance of their applications. Response time is one of the most important performance attributes for the web application. However, it is challenging to maintain and offer response time guarantees to the user of the web applications mainly due to the workload variability, performance varying cloud resources Adam et al. (2016); Ou et al. (2012), and unexpected user growth.

In order to offer a better quality of services (QoS), these cloud-hosted web applications need to maintain specific response time guarantees, which can be achieved by dynamic detection of bottlenecks and their effective resolution automatically. However, providing such response time could pose a great challenge to the cloud service providers mainly due to the inherent nature of multi-tier applications, which are typically complex and bottlenecks may occur on multiple locations depending upon the specific workload patterns at any given time interval.

Most of the Public Cloud service providers offer services to automatically scale web applications. However, these services do not support multi-tier applications to automatically scale all tiers dynamically on changing workload patterns. For example; Amazon Web Services (AWS)

CONTACT Waheed Iqba waheed.iqbal@pucit.edu.pk

Amazon.com, Inc. (2015) is a public cloud service provider, and whose Elastic Compute Cloud (EC2) Auto-scaling service, Amazon Inc. (2015) enables users to configure all their tiers to a horizontally scale base using user-defined policies and then users have to ensure connectivity configurations among tiers by themselves. In horizontal scaling, the application is scaled by adding more machines. Specifically, for a multi-tier web application, using EC2 Auto-scaling service, one can develop a custom auto-scaling service to scale the database and web tiers horizontally. The horizontal database scaling is quite challenging mainly due to the use of the master-slave architecture, which provides limited scaling only on read queries and introduces the additional overhead of data synchronization from master to slaves. However, horizontal scaling for the web server is appropriate as the multiple web servers does not have any dependency with each other and can interdependently serve the incoming workload. Vertical scaling is another technique, which is appealing for database services as it allows adding more power like CPU and RAM to allocate the machine dynamically. Therefore, using vertical scaling for the database server reduces data synchronization concerns and is a better solution compared to the horizontal scaling.

Currently, the multi-tier web applications are auto-scaled using machine learning Amiri & Mohammad-Khanli (2017); Gujarati et al. (2017); Bu et al. (2009) and heuristics Iqbal et al. (2011a); Han et al. (2014) based methods. These methods perform horizontal scaling on web tiers and mostly over-provision the database tier. There have been limited studies to investigate the combination of horizontal and vertical scaling together for multi-tier web applications. For example; Juan et al. Perez et al. (2018) studies the vertical and horizontal auto-scaling methods to reduce the application latency. However, this work does not provide a method to automatically scale both tiers using a combination of horizontal and vertical scaling together. Whereas, we advocate to use both the horizontal and vertical auto-scaling together for using multi-tier applications. Some of the existing work combines reactive and predictive auto-scaling methods and call them hybrid auto-scaling. A reactive auto-scaling method provision resource is based on a specific event, for example; on a specific CPU or response time threshold. Whereas, the predictive auto-scaling method initiates the resource provision with the anticipation of future workload needs. Anshuman et al., Biswas et al. (2017) propose an auto-scaling method, which initiates a horizontal scaling of web applications using a combination of reactive and predictive strategies. However, this work does not explore the use of vertical and horizontal scaling together.

In this paper, we propose and evaluate a new hybrid auto-scaling method to dynamically allocate the resources to a multi-tier web application hosted on the AWS to minimize violations of a specific response time requirement with minimal cost. Our proposed system to include the horizontally scale web server tier and vertical scale database tier dynamically offers better response time and overcomes the database horizontal scaling limitations. We evaluate our proposed auto-scaling using different configurations and compared it with traditional approaches including static and over-provisioning techniques to deploy the multi-tier web applications. In the static deployment, fixed infrastructure resources are allocated to the application, which cannot be scaled automatically. Whereas, in the overprovisioning technique, powerful resources are allocated to the application to avoid any saturation. To the best of our knowledge, this is the first study to use hybrid auto-scaling methods for multi-tier web applications hosted on the AWS to minimize violations of response time requirements with minimal cost. The main contributions of this paper include:

- A new hybrid auto-scaling method is proposed for dynamically scaling web server tier horizontally and database tier vertically.
- The proposed method is implemented on the AWS cloud.
- An extensive set of experiments are performed using a benchmark multi-tier web application and different settings of the proposed method.
- The proposed solution is compared with static and overprovisioning resource allocation methods.

There are a few limitations to this work. The web server and database are considered to be hosted on separate machines to work the proposed solutions effectively. Moreover, the proposed solution is designed explicitly for the two-tier web applications.

The rest of the paper is organized as follows: Related work is presented in Section 2. Our proposed hybrid auto-scaling method is explained in Section 3. Experimental design is presented in Section 4. Experimental results are given in Section 5. Finally, conclusion and future work are discussed in Section 6.

2 RELATED WORK

THERE have been several research efforts to manage multi-tier web application resources automatically. For example; Urgaonkar et al. in Urgaonkar et al. (2005) described a queuing network-based analytical model to learn the dynamics of each tier of the web application and dynamically allocate resources to prevent performance issues. Bonvin et al., Nicolas et al. (2011) provided a cost-effective solution for the dynamic provisioning of cloud resources to the multi-tier web applications in order to satisfy the response time and availability guarantees using horizontal scaling.

Most of the research in dynamic resource provisioning of web applications, Villela et al. (2007); Bodik et al. (2009); Dejun et al. (2011) have been focused on the provision of more resources as the workloads increase in order to maintain the applications performance. Some of the researchers have used machine learning techniques to learn workload patterns of multi-tier web applications to provision resources automatically. For example; Singh et al. in Singh et al. (2010) have presented a technique to model dynamic workloads for multi-tier Web applications using k-means clustering on the service time feature that collect logs at each tier. The method uses queuing theory to model the system's reaction to the workload and identifies the number of Amazon EC2 instances required that helps to maintain the performance of the web application under the given workload. Gemma Reig et al. Reig & Guitart (2012) proposed a system to predict future demand of the CPU by combining machine learning and statistical techniques to maintain the QoS in dynamic workloads for web applications.

Daniel Villela et al. Villela et al. (2007) have proposed a model for scaling of only the application tier in a typical multi-tier application. The authors have analyzed the actual trace of arriving requests on the application tier of an e-commerce website and derived methods to approximate the resource allocation to reduce cost by modeling servers of application tier as the M/G/1/PS queuing system. Similarly, Peter et al. in Bodik et al. (2009), proposed a statistical machine learning based model that claims to address the shortcomings of various models used in existing data-centers. The authors have argued that by using analysis, control and modeling techniques of statistical machine learning, one can fix the shortcomings common in a data-center. Jiang Dejun et al., Dejun et al. (2011), have studied the resource heterogeneity of cloud infrastructure, load balancing and tier selection for dynamic resource provisioning of the cloud-hosted multi-tier web applications. The authors have shown that identical provisioned resources perform significantly different in performance. Therefore, balancing an equal amount of load among each provisioned resource will lead to poor resource utilization and performance, even if the resources look identical. Secondly, the dynamic provisioning of resources need to show careful identification of the tier to avoid any possible SLA violations, so that the newly added instance can be properly utilized.

In Yazdanov & Fetzer (2014), the authors have purposed a vertical scaling strategy using the reinforcement learning method for a local testbed cloud by controlling allocations of the CPU and memory to the virtual machine hosting only using the application tiers. Similarly, in Iqbal et al. (2015), the authors have performed the horizontal scaling of a multi-tier web application hosted on the Amazon Web Service (AWS) using coarse-grained access log monitoring techniques. Hector Fernandex et al., Fernandez et al. (2014a) purposed an auto-scaling system that exploits resource heterogeneity, and provides load balancing according to the capacity of resources and define multiple levels of QoS agreements (metal classification) to balance the SLA fulfillment and provisioning cost. They have defined a system to profile every resource to measure the capacity and provide weighted load balancing. F. Seracini et al. in Seracini et al. (2014) exploited the resource planner (EcoWare) along with some modifications that takes the responsibility of identifying the correct amount of resources by notifying the resource planner when the SLA violation occurs. Whenever the system detects the need of an increase in the provisioned resources, it invokes the resource planner algorithm that first checks the availability of the server's in the free server's pool and increases the number of allocated servers until the SLA is satisfied. Hector Fernandez et al. in Fernandez et al. (2014b) argued that there are three main reasons that limit the use of the sophisticated techniques proposed in various academic researches that provide much less gain than the effort required, and implementation is difficult and show unrealistic evaluations, because of the fabricated workloads. The authors claimed that they investigate real problems and they propose simple solutions that provide better performance without increasing the complexity overhead. Their proposed techniques set different threshold levels for the prediction of performance degradation that may occur in the future, so that proper resource requirement can be calculated. Also, to handle the resource heterogeneity, it provides dynamic weights for the load balancing.

In Qu et al. (2016), the authors proposed a cost-effective resource scaling algorithm for the web applications using heterogeneous AWS spot instances. The main contribution of the paper was to use appropriate policies to exploit the spot instances to reduce the cost of web applications. Authors in Grimaldi et al. (2017) proposed and evaluated a fuzzy approach to manage the allocated virtual machines to an application running on the AWS cloud. The work only exploits horizontal scaling. Recently auto-scaling applications are addressed by many researchers. For example; the authors in Krieger et al. (2017) provide the auto-scaling method for bioinformatics and biomedical applications. In Satoh (2016) the authors provide a system to adaptively manage the resources for mobile applications. The authors in Khoshkbarforoushha et al. (2016) proposed and evaluated the dynamic resource management system for the big data stream analytics applications.

A comprehensive review of the auto-scaling strategies is conducted in Lorido-Botran et al. (2014), this review divides the auto-scaling strategies into five categories to analyze pros and cons of each strategy.

This work also highlights that distribution and replication of databases, prunes additional issues and a few work efforts are conducted to deal with the issues of database scaling. In Papadopoulos et al. (2016), the authors presented a framework to evaluate the auto-scaling strategies using simulations. The authors used six different auto-scaling strategies to evaluate their proposed framework. Anju and Inderveer Bala & Chana (2016) proposed and evaluated a proactive load balancing approach based on the VM migrations using simulations.

There have been limited studies to investigate the combination of horizontal and vertical scaling together for the multi-tier web applications. For example; recent work done by Juan et al., Perez´ et al. (2018) study the hybrid auto-scaling combining the vertical and horizontal for multi-tier web applications. However, this work does not provide a method to automatically scale both tiers using a combination of horizontal and vertical scaling together.

It is worth mentioning that none of the above-mentioned works explore a possibility to combine the horizontal and vertical scaling together to provision resources to different tiers of a multi-tier web application hosted on a public cloud. The research work reported in this paper is a significant extension of our preliminary work Nisar et al. (2015). This paper presents a comprehensive evaluation using the improved methodology and a new set of experiments to evaluate the performance and cost-effectiveness of our proposed hybrid auto-scaling method with the comparison to existing industry practicing methods.

3 PROPOSED HYBRID AUTO-SCALING OF MULTI-TIER WEB APPLICATIONS

IN this section, we explain our proposed hybrid auto-scaling approach for multi-tier web applications hosted on the Amazon Web Service (AWS) cloud architecture. We consider a simple two-tier web application consisting of a web server (web tier) and a database server (database tier) to dynamic provisioning of the AWS resources on varying workloads. We assume that each tier is deployed on a separate virtual machine (instance). Our proposed approach, scale-out (horizontal scaling) web tier and scale-up (vertical scaling) database tier is used whenever a bottleneck on a specific tier is detected. The bottleneck on a specific tier is detected using a black-box approach similar to Iqbal et al. (2011b), which monitors the application response time and CPU utilization to identify the bottleneck. Whenever the CPU utilization or response time reaches a specific threshold, the system announces a bottleneck point.

Let W denote a set of n virtual machine instances provisioned to the web server tier. Where $W_j \in$ W is the specific j^{th} provisioned virtual instance in the web server tier. Each j^{th} web server instance is associated with the specific properties as $<W_{type_j}, W_{cpu_j}, W_{cpu_{j,k\ utl}}>$, where:

W^{type}_j = type of j^{th} web tier instance.

W^{cpu}_j = total CPU cores of the j^{th} web tier instance.

$W^{cpu}_{j,k\ \ utl}$ = the CPU utilization of the j^{th} web tier instance at the k^{th} time interval.

For the database tier, let D denote a set of virtual machine instances available for provisioning at any time interval. Where $D_i \in$ D, represents the currently provisioned virtual instance.

The provisioned database tier instance is characterized as $<Dtype_i, Dcpu_i, Dcpu_{i,k}\ utl>$, where:

D^{type}_i = type of the l^{th} database tier instance.

D^{cpu}_i = total CPU cores of the i^{th} database tier instance.

$D^{cpu}_{i,k\ \ utl}$ = current CPU utilization of the i^{th} database tier instance at the k^{th} time interval.

3.1 Web Tier Horizontal Auto-scaling

Algorithm 1 explains the web tier auto-scaling method. The web tier of the application is deployed on an EC2 instance. We configure the AWS auto-scaling service in such a way that it increases the number of virtual machines allocated to the web tier whenever the average CPU utilization of allocated EC2 instances crosses the user defined upper CPU utilization threshold (C_u) for specific p consecutive last time intervals. We define t as the unit of the time interval to wait before profiling virtual instances. Similarly, we define a policy in the the AWS auto-scaling service to decrease the number of allocated virtual machines whenever the average CPU utilization of all allocated EC2 instances stay less than a user-defined lower CPU threshold (U_l) for specific p consecutive last time intervals.

These parameters can be varied to test different settings. For example, setting the higher value of C_u will slow down the scale up decision as it will keep waiting for the CPU utilization to reach the higher level. However, setting the higher value of C_l will enable the system to quickly reduce the allocated resources as the system will start to scale down as soon as the CPU utilization reaches to the lower level of the CPU utilization. In our experimental evaluation, we used upper CPU utilization threshold C_u = 70%, lower CPU utilization threshold C_l = 30%, last profiling time intervals count p = 30, and time interval t = 20 seconds for the web tier horizontal auto-scaling.

3.2 Database Tier Vertical Auto-scaling

We explain our proposed algorithm to the vertical scale database tier automatically in Algorithm 2. We pre-defined a set of database instances D from which one of the instance $D_i \in$ D, where $1 \leq i \geq max$, can be provision as a database tier dynamically. The instance D_1 is the weakest instance where D_{max} is the most powerful instance available in D in terms of the CPU, memory, I/O, and the bandwidth resources.

Algorithm 1: WEB TIER AUTO SCALING ALGORITHM

Input: t (profiling time interval in seconds), C_u (upper CPU utilization threshold in percentage), C_l (lower CPU utilization threshold in percentage), and p (number of last profiling intervals to compute the CPU utilization of the virtual instances).

Result: Updated $\mathbb{W}$, it would be either increase, decrease, or remain unchanged.

```
foreach t interval
begin
    if ⟨ (∀k∈p Σ_{j=1}^{|W|} W_{j,k}^{cpu_util}) / (|W|*p) ⟩ ≥ C_u then
        W^+        // increase the number of virtual machines
    end
    else if ⟨ (∀k∈p Σ_{j=1}^{|W|} W_{j,k}^{cpu_util}) / (|W|*p) ⟩ ≤ C_l and |W| ≠ 1 then
        W^-        // decrease the number of virtual machines
    end
end
```

Algorithm 2: DATABASE TIER SCALING ALGORITHM

Input: t (profiling time interval in seconds), C_u (upper CPU utilization threshold in percentage), C_l (lower CPU utilization threshold in percentage), S_m (scale mode, it can either be RAPID or GRADUAL. For RAPID, the most powerful instance will be used whereas for GRADUAL the database is upgraded stepwise), and p (number of last profiling intervals to compute the CPU utilization of the virtual instances).

Result: Upgrade, downgrade, or remain the same instance used for the database tier.

```
foreach t interval
begin
    if ⟨ (∀k∈p Σ D_i^{cpu_util}) / p ⟩ ≥ C_u and D_i ≠ D_max then
        if S_m = RAPID then
            provision D_max ∈ 𝔻
            release D_i ∈ 𝔻
            i ← max
        end
        else
            provision D_{i+1} ∈ 𝔻
            release D_i ∈ 𝔻
            i ← i + 1
        end
    end
    else if ⟨ (∀k∈p Σ D_i^{cpu_util}) / p ⟩ ≤ C_l and i ≠ 1 then
        if S_m = RAPID then
            provision D_1 ∈ 𝔻
            release D_i ∈ 𝔻
            i ← 1
        end
        else
            provision D_{i-1} ∈ 𝔻
            release D_i ∈ 𝔻
            i ← i - 1
        end
    end
end
```

We develop a database vertical scaling service in such a way, that dynamically upgrades the database instance whenever the average CPU utilization of the allocated instances crosses the user defined upper CPU utilization threshold (C_u) for a specific p consecutive last time interval. Similarly, the proposed vertical scaling service downgraded the allocated virtual machine whenever the average CPU utilization of the allocated virtual instance stays less than a user-defined lower CPU threshold (C_l) for a specific p consecutive last time interval. The algorithm also accepts an input namely scale mode (S_m). This setting can be either RAPID or GRADUAL. If $S_m = RAPID$ then whenever the upper CPU utilization threshold (C_u) observed the system automatically will allocate the most powerful available instance D_{max} to the database tier. However, if $S_m = GRADUAL$ then the database tier is upgraded stepwise to the next level. For scale down, if $S_m = RAPID$ is used then the database tier dynamically is downgraded to the minimum possible allocation. Similarly, if $S_m = GRADUAL$ then the database tier was downgraded stepwise to the previous level.

In our experimental evaluation, we used different types of EC2 and RDS instances for the database vertical scaling. We used $C_u = 70\%$, $C_l = 20\%$, $p = 30$, and $t = 20$ seconds. The reason we choose $C_u = 70$ is to provide sufficient time to the system for handling unexpected workloads, which increases the CPU usage and also it is a reasonable threshold for efficient utilization of the server resources Allspaw (2008). Where $C_l = 20$ is used to scale down the resources as it indicates the resources are underutilized and the system can bear to release some resources. The reasons to use $p = 30$ is to capture the relatively long term behavior of the resources. However, $t = 20$ seconds is used to define the interval duration to monitor and profile the application performance, the lower this value will allow the auto-scaler to quickly identify the need of scaling and higher value of this will slow down the auto-scaling decision. We perform different experiments using both possible values of the scale mode S_m.

4 EXPERIMENTAL DESIGN

IN this section, we explain the experimental benchmark web application, synthetic workload generation method, and experiments performed to evaluate our proposed hybrid auto-scaling approach.

4.1 Benchmark Multi-tier Web Application

We have used the RUBiS OW2 Consortium (1999), an open-source web application, which provides an auction facility of items to include bidding, selling, and browsing of items similar to eBay, and as a benchmark application to evaluate hybrid auto-scaling methods. The RUBiS application provides the buyer, visitor, and seller as three different user roles to interact with the application. The users with either the buyer or seller role needs to register with the application before using it. However, the users with a visitor's role can use the application mainly to browse the available items without registration.

The RUBiS is widely used in web application autonomous management research. We have used the PHP implementation of the RUBiS with the MySQL database in our experimental evaluation.

4.2 Workload Generation

We have used the Apache JMeter Apache Software Foundation (1999) to generate synthetic workloads for the RUBiS benchmark web application in each of the experiments. The Apache JMeter Apache Software Foundation (1999); Halili (2008) is an open-source tool written in Java to generate a synthetic workload for the web applications to test the performance and behavior of the application under varying workloads. Our synthetic workload generation emulates a specific number of concurrent user sessions per second for the RUBiS web application in a step-up fashion. Each user session issues a read requests to Categories, Regions, Items, Home, Register, and Sell pages of the RUBiS. We have configured the JMeter with 750 number of threads (users) ramping in a period of 1200 seconds.

We have performed six experiments based on this workload for the RUBiS. Experiment 1 profiles the systems behavior under the static allocation. Experiment 2 profiles the systems behavior under the over-provisioning of the database tier and a dynamic scaling of the web tier. In Experiments 3, 4, 5, and 6, we profile our proposed hybrid auto-scaling scheme under the different scale mode and instance types. We generated the workload for each experiment for a total of 70 minutes, which contains the ramp-up period of the first 55 minutes and then the workload starts decreasing rapidly until the end of each experiment.

4.3 Experiment Details

Figure 1 explains the deployment of our proposed hybrid auto-scaling for a multi-tier web application using the Amazon Web Services (AWS). In a typical scenario, the workload is received by the Elastic Load Balancing (ELB) that distributes it to the allocated web tier instances. Then the web tier instances may query the provisioned instance of the database tier to generate the response. The diagram shows only the RDS instances, however, we also evaluate the proposed methodology using a different type of the EC2 instances for the database tier. We performed six different experiments to evaluate the performance and cost of our proposed hybrid auto-scaling method. Table 1 summarizes these experiments and Table 2 describes the resource allocation of the different instances used during the experiments.

In Experiment 1, the static allocation (St-Al), we have installed the Apache web server on the Amazon EC2 instance of the type m3.medium and deployed the RUBiS benchmark web application (web tier) on it. We have used the Amazon RDS instance of the type db.m3.medium (M) configured with the MySQL and deployed the RUBiS data (DB tier) on it. We did not enable any kind of dynamic scaling for both tiers in Experiment 1.

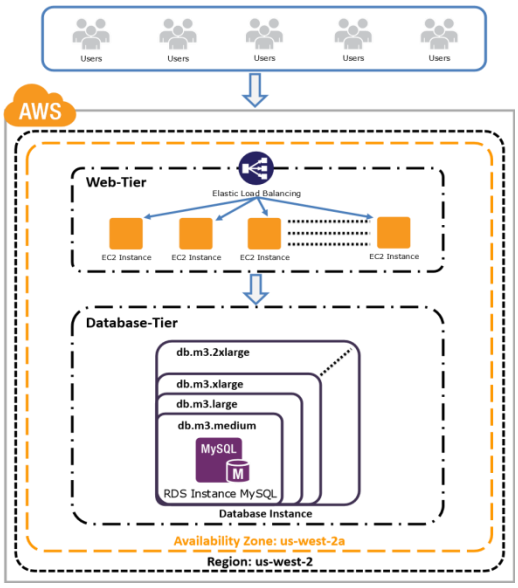


Figure 1. **The Proposed Hybrid Auto-scaling Combining the Horizontal and Vertical Scaling for the Multi-tier Web Applications using the Amazon Web Services.**

In Experiment 2, for the over-provisioning database tier (Op-RDS), we enabled the horizontal scaling on the web tier and over-provisioned the database tier using the Amazon RDS instance of the type db.m3.2xlarge (2XL).

Table 1. **Summary of the Conducted Experiments.**

Experiment	Description
1: St-Al	The static allocation is using one Amazon EC2 instance of the type m3.medium for the Web server tier and one RDS instance of the type db.m3.medium for the database tier.
2: Op-RDS	The horizontal auto scaling is enabled for the web server tier and the over-provisioned database tier using a powerful Amazon RDS instance of the type db.m3.2xlarge.
3: Gr-RDS	The horizontal auto scaling is enabled for the web server tier and the *vertical scaling* using the *RDS instances* with the S_m = GRADUAL setting for the database tier.

Experiment	Description
4: Ra-RDS	The horizontal auto scaling is enabled for the web server tier and the vertical scaling using the RDS instances with the S_m = the RAPID setting for the database tier.
5:Gr-EC2	The horizontal auto scaling is enabled for the web server tier and the vertical scaling using the EC2 instances with the S_m = the GRADUAL for the database tier.
6:Gr-EC2-Pb	The horizontal auto scaling is enabled for the web server tier and vertical scaling using the EC2 instances with the S_m = the GRADUAL and an extra pre-booted next level instance ready to quickly upgrade the database tier.

In Experiment 3, using both the gradual scaling database tier and the RDS instance (Gr-RDS), we enabled the horizontal scaling on the web tier and enabled the vertical scaling on the database tier. For the vertical database scaling, we use the S_m = *GRADUAL* setting in the proposed database scaling algorithm. It enables the auto-scaling service to use the instance type's db.m3.medium CPU and threshold (C_u) is observed, the system automatically upgrades the database tier to the next level. Whenever the lower CPU threshold (C_l) is observed, the service automatically downgraded the database tier to the previous level.

In Experiment 4, the rapid scaling database tier using the RDS instance (Ra-RDS) is used. We enabled the horizontal scaling on the web tier and enabled the vertical scaling on the database tier. For the vertical database scaling, we use the S_m = *RAPID* setting in the proposed database scaling algorithm. It enables the auto-scaling service to use either the db.m3.medium (M) or the db.m3.2xlarge (2XL) type of RDS instance dynamically for the database tier. Initially, the db.m3.medium is provisioned to the database tier and whenever the upper CPU threshold (C_u) is observed, the auto-scaling service automatically upgrades the database tier to the maximum level of (2XL). Similarity, whenever the lower CPU threshold (C_l) is observed, the auto-scaling service automatically downgraded the database tier to the minimum allocation of level(M).

Experiment 5, the gradual scaling database tier using the EC2 instance (Gr-EC2), is similar to Experiment 3; however, we used the EC2 instances instead of the RDS instances to dynamically upscale the database tier. The database auto-scaling service can dynamically switch instance types using one of the EC2 instances of the type m3.medium (M), the m3.large (L), the m3.xlarge (XL), and the m3.2xlarge (2XL). Initially, the m3.medium is provisioned to the database tier and whenever the upper CPU threshold (C_u) is observed, the auto-scaling service automatically upgrades the database tier to the next level. Similarity, whenever the lower CPU threshold (C_l) is observed, the auto-scaling service automatically downgraded the database tier to the previous level

Experiment 6, the gradual scaling database tier using the pre-booted EC2 instance (Gr-EC2-Pb), is similar to Experiment 5, however, we always keep an extra pre-booted next level EC2 instance ready to be quickly upgraded the database tier.

5 EXPERIMENTAL RESULTS

5.1 Experiment 1: Static Allocation (St-Al)

THIS section explains the results we obtained in Experiment 1. Figure 2 shows the average throughput (requests/second) and the average response time of the application during Experiment 1. The web and database tiers are statically allocated using the m3.medium and the db.m3.medium type of instances respectively at the beginning of the experiment. After the 22nd minute of the experiment, the response time of the application starts growing dramatically and crosses acceptable response time threshold. However, the response time starts decreasing after the 62nd minute as the amount of the workload has started to be reduced by the workload generator.

This experiment provides the baseline performance and depicts the systems behavior under the traditional deployment of a multi-tier web application. Once the CPU resources of either the web tier or database tier saturate, then the throughput of the application stops growing, the response time starts increasing, and the number of requests processed by the servers starts decreasing.

Table 2. The Instance Type and Resource Allocation of the Instances used in the Experiments.

Instance Type	vCPU	Memory (GiB)	Network (Perf)	Cost /Hour
m3.medium	1	3.75	Moderate	0.067
db.m3.medium (M)	1	3.75	Moderate	0.09
db.m3.large (L)	2	7.5	Moderate	0.785
db.m3.xlarge (XL)	4	15	High	0.37
db.m3.2xlarge (2XL)	8	30	High	0.74
m3.medium (M)	1	3.75	Moderate	0.067
m3.large (L)	2	7.5	Moderate	0.133
m3.xlarge (XL)	4	15	High	0.266
m3.2xlarge (2XL)	8	30	High	0.532

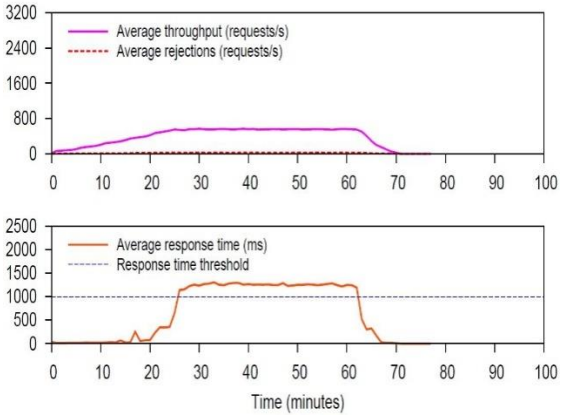


Figure 2. The Average Response Time, Throughput, and Rejections in Experiment 1.

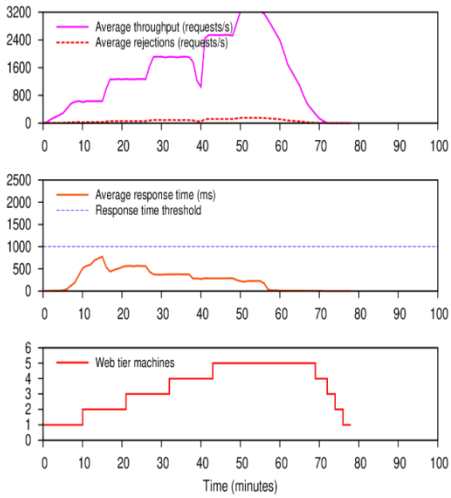


Figure 3. The Average Response Time, Throughput, Rejections, and Dynamic Allocation of Instances to the Web Tier in Experiment 2. The Web Tier Horizontal Scales on the CPU and Saturation of the Allocated Instances. The Database Tier is Over-provisioned by using the db.m3.2xlarge RDS Instance.

5.2 *Experiment 2: Over-provisioning the Database Tier (Op-RDS)*

Figure 3 shows the average response time, the average throughput, the average rejections, and the dynamic provisioning of the web tier instances during Experiment 2. The number of instances allocated to the web tier is dynamically managed using the approach explained in 3.1. The database tier is over-provisioned by using the db.m3.2xlarge type of instance. During this experiment, the average response time never violates the response time threshold and the throughput increases gradually until the 55^{th} minute of the experiment. After this, the throughput decreases due to the workload and the generator starts reducing the number of user requests after the 55^{th} minute.

This experiment provides baseline performance with overprovisioned resources to ensure the system performs well on unexpected workloads. One can argue that this approach provides better performance than the rest of the schemes, however, adopting this scheme, which is over-provisioning of the database tier would be expensive for the application owners and therefore is not a cost-effective solution.

5.3 *Experiment 3: The Gradual Scaling Database Tier using the RDS Instances (Gr-RDS)*

Figure 4 shows the average response time, the average throughput, the average rejections, and the provisioning of the web and database tier during Experiment 3. The number of instances allocated using the GRADUAL configuration for the varying workload. It can be seen from the graph that our proposed hybrid auto-scaling method appropriately brings down the response time by automatically adjusting the web and database tier resources.

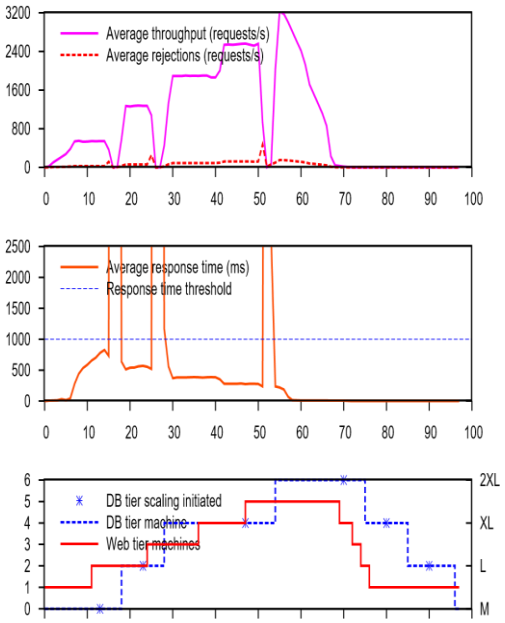


Figure 4. The Average Response Time, Throughput, Rejections, and Dynamic Allocation of the Web Tier and the Database Tier in Experiment 3. The Bottom Graph's y2 Title used to Show the Database Tier Allocation during the Experiment. The Bottom Graph also shows the Dynamic Addition of the Web Tier and the Dynamic Switching of the Database Tier Instance.

5.4 *Experiment 4: The Rapid Scaling Database Tier using the RDS Instance (Ra-RDS)*

Figure 5 shows the average response time, the average throughput, the average rejections, and the provisioning of the web and database tier during Experiment 4. The number of instances allocated to the web tier is dynamically managed using the approach explained in Section 3.1. The database tier instance type is dynamically managed using the approach explained in Section 3.2 with the S_m = *RAPID* configuration. It can be seen that the database scaling strategy rapidly changes the RDS instance from the db.m3.medium (M) to the db.m3.2xlarge (2XL). This is an effective strategy to rapidly upgrade the database allocation to accommodate huge workload variations effectively.

5.5 *Experiment 5: The Gradual Scaling Database Tier using the EC2 Instance (Gr-EC2)*

Experiment 5 is similar to Experiment 3 except we used the EC2 instances instead of the RDS instances for the vertical scaling of the database tier. The number of instances allocated to the web tier is dynamically managed using the approach described in Section 3.1. The database tier instance type is dynamically managed using the approach explained in Section 3.2 with the S_m = *GRADUAL* configuration. The experiment results in Figure 6 show the effectiveness of our proposed hybrid auto-scaling method using the GRADUAL configuration for the varying workload. It can be seen from the graph that our proposed hybrid auto-scaling method appropriately brings down the response time by automatically adjusting the web and database tier resources. The number of overall processed requests are compared more to Experiment 3, however, during the operation of database scaling the number of rejections increase.

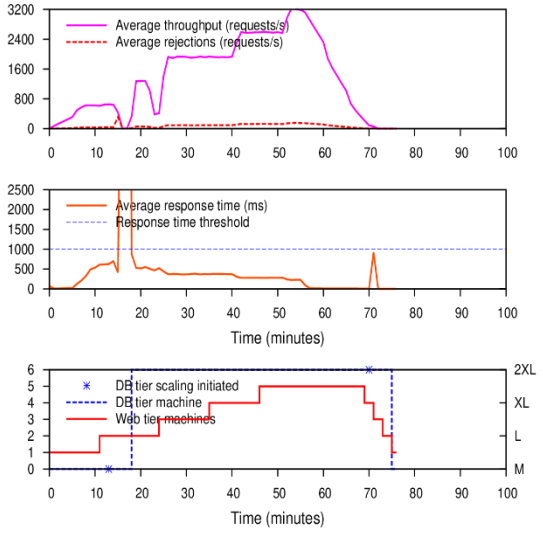


Figure 5. The Average Response Time, Throughput, Rejections, and Dynamic Allocation of the Web Tier and the Database Tier in Experiment 4.

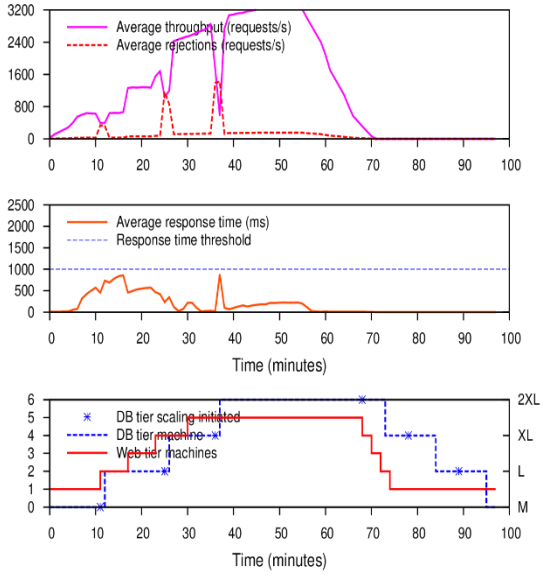


Figure 6. The Average Response Time, Throughput, Rejections, and Dynamic Allocation of the Web Tier and the Database Tier in Experiment 5.

5.6 Experiment 6: The Gradual Scaling Database Tier using the Pre-booted EC2 Instance (GR-EC2-PB)

Figure 7 shows the average response time, the average throughput, the average rejections, and the provisioning of the web and database tier during Experiment 6. The number of instances allocated to the web tier is dynamically managed using the approach explained in Section 3.1. The database tier configured on the Amazon EC2 instance is dynamically managed using the database vertical scaling approach explained in Section 3.2. However, instead of booting the next level instance for the database tier, we always keep an extra pre-booted EC2 instance of the next level to reduce the time to launch the new instance. It can be observed from Figure 7 that using the pre-booted EC2 instance dramatically reduces the time of switching among the instances approximately one minute along with a slight increase in rejections.

5.7 Experimental Summary

We summarized the experimental results in Table 3. It shows that the total number of requests processed (Processed); the total number of requests unprocessed or rejected (Unprocessed), the percentage of requests missing the acceptable response time threshold (SLA misses), the number of web tier scale operations (Web scales), number of database tier scale operations (DB Scales), and the total cost (Cost) for each experiment. We measured the cost of each experiment by summing up the cost of the resources used during each of the experiments. The total requests generated by the workload generator in each of the experiments were 8.26 million.

Table 3. The Experimental Results Summary.

Experiment	Processed (millions)	Unprocessed (millions)	SLA misses (%)	Web scales (#)	DB scales (#)	Cost (USD)
1: **St-Al**	1.760	6.500	37.71	0	0	$0.314
2: **Op-RDS**	6.685	1.575	05.80	8	0	$1.167
3: **Gr-RDS**	5.512	2.748	06.81	8	6	$0.784
4: **Ra-RDS**	6.468	1.792	06.20	8	2	$1.007
5: **Gr-EC2**	7.472	0.788	11.11	8	6	$0.732
6: **Gr-EC2-Pb**	7.459	0.801	09.85	8	6	$0.917

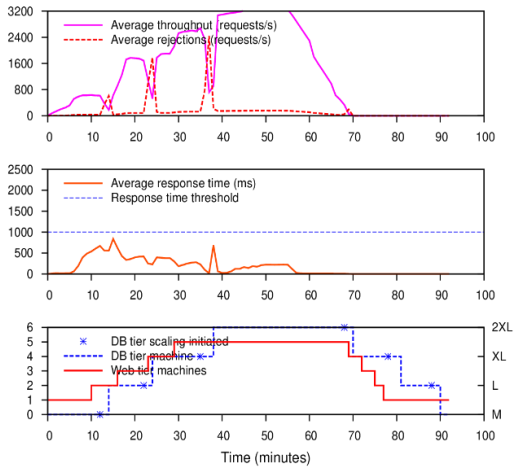


Figure 7. The Average Response Time, Throughput, Rejections, and Dynamic Allocation of the Web Tier and Database Tier in Experiment 6. The Bottom Graph's y2 Title used to Show the Database Tier Allocation during the Experiment.

It appears from the table that the over-provisioning of the database tier (Experiment 2) outperformed the other experiments in terms of performance; however, this is not a cost-effective solution and also shows a larger number of unprocessed requests. The proposed hybrid auto-scaling method with gradual scaling using both of the RDS and EC2 instances (Experiment 3 and 4) reduced the cost significantly. The maximum number of requests is also processed using the gradual setting with the EC2 instances (Experiment 5).

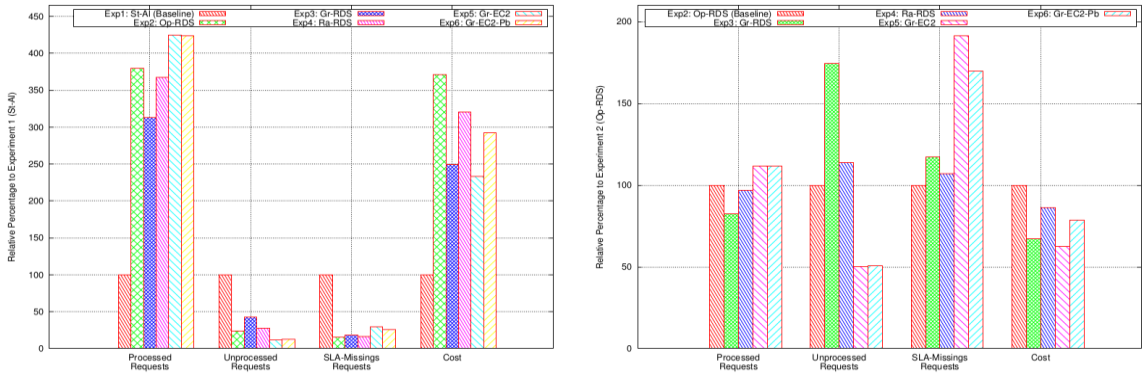
To compare the results obtained using the proposed auto-scaling method (Experiments 3, 4, 5, and 6) we used Experiment 1 and Experiment 2 as baseline methods. Figure 8(a) shows the relative percentage of processed requests, unprocessed requests, SLA missing requests, and cost in comparison with Experiment 1 (static allocation) and Figure 8(b) shows the comparison with Experiment 2 (over-provisioned resources). The relative percentages for each of the evaluation metrics for the proposed methods are computed by considering the baseline method, which yields 100%. Figure 8(a) shows that the proposed method using all different settings outperform the static allocation results significantly for the processed requests, unprocessed requests, and the SLA-missing requests. However, the cost of the static allocation is significantly lower. Figure 8(b) shows that the proposed method using the gradual scaling with the EC2 instances (Experiment 5 and 6) outperforms the number of processed requests, number of unprocessed requests, and cost comparing to Experiment 2. However, results obtained in Experiments 5 and 6

show higher SLA-missing requests compared to Experiment 2.

The proposed hybrid auto-scaling method can help owners of the multi-tier web applications to use either the RDS instances or the EC2 instances to vertically scale the database tier; however, the application can still horizontally scale the web server tier. We strongly believe that the proposed method provides significant cost benefits to the multi-tier application owners with an acceptable performance compared to the traditional static and over-provisioning methods. Our extensive evaluation of using different settings of the proposed method allows users to enable the appropriate settings to achieve specific goals. For example, if an application owner wants to minimize the unprocessed/rejected requests with the minimal cost, then the most feasible setting for the owner is to enable the gradual scaling with the EC2 instances, similar to Experiment 5.

6 CONCLUSION AND FUTURE WORK

IN this paper, a new hybrid auto-scaling method to dynamically provision resources to the multi-tier web applications hosted on the Amazon public cloud was proposed and evaluated. We showed that the proposed auto-scaling method dynamically scales the web server tier horizontally and scales the database tier vertically in order to provide acceptable response time on varying workloads to be cost effective. We have demonstrated that our experimental evaluation validates the effectiveness of our proposed method compared to the current practices of static and overprovisioning of the database tier. To the best of our knowledge, our proposed method is one of the first attempts to propose and evaluate a hybrid auto-scaling method for the multi-tier web applications hosted on public clouds, which dynamically uses horizontal and vertical scaling for different tiers.



(a) Comparison of all experiments with Experiment 1

(b) Comparison of all experiments with Experiment 2

Figure 8. The Experimental Results Comparison, Processed Requests, Unprocessed Requests, SLA Missing Requests, and Cost are Computed Relative to the Baseline.

Currently, we are extending our hybrid auto-scaling method to support the generic n-tier application architectures and also incorporate proactive scaling decisions to minimize the percentage of the SLA missing requests.

7 REFERENCES

Abdullah, M., Khana, S., Alenezi, M., Almustafa, K., & Iqbal, W. (2018). Application Centric Virtual Machine Placements to Minimize Bandwidth Utilization in Datacenters. *Intelligent Automation and Soft Computing, 1–14. doi:10.31209/2018.100000047*

Adam, O. Y., Lee, Y. C., & Zomaya, A. Y. (2016). Constructing performance predictable clusters with performance-varying resources of clouds. *IEEE Transactions on Computers*, *65*, 2709–2724.

Allspaw, J. (2008). *The art of capacity planning: scaling web resources.* " O'Reilly Media, Inc.".

Amazon Inc. (2015). Amazon Web Services auto scaling. Available at https://aws.amazon.com/auto scaling/ [Online; accessed 6-Nov-2015].

Amazon.com, Inc. (2015). Amazon Web Services (AWS). Available at https: //aws.amazon.com/ [Online; accessed 6-Nov-2015].

Amiri, M., & Mohammad-Khanli, L. (2017). Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, *82*, 93–113.

Apache Software Foundation (1999).Apache JMeter. http://jmeter. apache.org/.

Bala, A., & Chana, I. (2016). Prediction-based proactive load balancing approach through VM migration. *Engineering with Computers*, *32*, 581–592.

Biswas, A., Majumdar, S., Nandy, B., & El-Haraki, A. (2017). A hybrid auto-scaling technique for clouds processing applications with service level agreements. *Journal of Cloud Computing*, *6*, 29.

Bodik, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., & Patterson, D. (2009). Statistical machine learning makes automatic control practical for internet datacenters. In *HotCloud'09: Proceedings of the Workshop on Hot Topics in Cloud Computing*.

Bu, X., Rao, J., & Xu, C.-Z. (2009). A reinforcement learning approach to online web systems auto-configuration. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems* ICDCS '09 (pp. 2–11). IEEE Computer Society.

Dejun, J., Pierre, G., & Chi, C.-H. (2011). Resource provisioning of Web applications in heterogeneous clouds. In *Proceedings of the 2nd USENIX Conference on Web Application Development*.

Fernandez, H., Pierre, G., & Kielmann, T. (2014a). Autoscaling Web Applications in Heterogeneous Cloud Infrastructures. In IEEE (Ed.), *IEEE International Conference on Cloud Engineering*. Boston, MA, Etats-Unis.´

Fernandez, H., Stratan, C., & Pierre, G. (2014b). Robust Performance Control for Web Applications in the Cloud. In *4th International Conference on Cloud Computing and Services Science*. Barcelona, Espagne. Best paper award.

Grimaldi, D., Pescape, A., Salvi, A., Persico, V. et al. (2017). A fuzzy approach based on heterogeneous metrics for scaling out public clouds. *IEEE Transactions on Parallel and Distributed Systems*.

Gujarati, A., Elnikety, S., He, Y., McKinley, K. S., & Brandenburg, B. B. (2017). Swayam: distributed auto scaling to meet slas of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (pp. 109–120). ACM.

Halili, E. H. (2008). *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packet Publishing Ltd.

Han, R., Ghanem, M. M., Guo, L., Guo, Y., & Osmond, M. (2014). Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Computer Systems*, *32*, 82–98.

Iqbal, W., Dailey, M., & Carrera, D. (2015). Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multi-tier web applications. In *Systems Journal, IEEE* (pp. 1–12). IEEE.

Iqbal, W., Dailey, M. N., Carrera, D., & Janecek, P. (2011a). Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, *27*, 871–879.

Iqbal, W., Dailey, M. N., Carrera, D., & Janecek, P. (2011b). Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, *27*, 871–879.

Khoshkbarforoushha, A., Khosravian, A., & Ranjan, R. (2016). Elasticity management of streaming data analytics flows on clouds. *Journal of Computer and System Sciences, In Press*, –.

Krieger, M. T., Torreno, O., Trelles, O., & Kranzlmuller, D. (2017). Building¨ an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows. *Future Generation Computer Systems*, *67*, 329–340.

Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, *12*, 559–592.

Nicolas, B., Thanasis G., P., & Karl, A. (2011). Automatic SLA-driven provisioning for cloud applications. In *Proceedings of the 2011 International Symposium Cluster, Cloud and Grid Computing*. Newport Beach, CA, USA: IEEE Computer Society.

Nisar, A., Iqbal, W., Bokhari, F. S., & Bukhari, F. (2015). Hybrid auto-scaling of multi-tier web applications: A case of using amazon public cloud. In *4th International Conference on Internet Applications, Protocols and Services* (pp. 274–279). Cyberjaya, Malaysia.

Ou, Z., Zhuang, H., Nurminen, J. K., Yla-J¨ a¨aski, A., & Hui, P. (2012). Exploiting hardware heterogeneity within the same instance type of amazon ec2. In *HotCloud*.

OW2 Consortium (1999). RUBiS: An auction site prototype. http://rubis. ow2.org/.

Papadopoulos, A. V., Ali-Eldin, A., Årzen, K.-E., Tordsson, J., & Elmroth, E.´ (2016). Peas: A performance evaluation framework for auto-scaling strategies in cloud applications. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, *1*, 15.

Perez, J. F., Chen, L. Y., Villari, M., & Ranjan, R. (2018). Holistic workload´ scaling: A new approach to compute acceleration in the cloud. *IEEE Cloud Computing*, *5*, 20–30.

Qu, C., Calheiros, R. N., & Buyya, R. (2016). A reliable and cost-efficient auto scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications*, *65*, 167–180.

Reig, G., & Guitart, J. (2012). On the anticipation of resource demands to fulfill the qos of saas web applications. In *ACM/IEEE 13th International Conference on Grid Computing* (pp. 147–154). ACM/IEEE ACM/IEEE.

Satoh, I. (2016). Self-adaptively auto-scaling for mobile cloud applications. *Procedia Computer Science*, *94*, 9–16.

Seracini, F., Menarini, M., Krueger, I., Baresi, L., Guinea, S., & Quattrocchi, G. (2014). A comprehensive resource management solution for

web-based systems. In *11th International Conference on Autonomic Computing (ICAC 14)* (pp. 233–239). Philadelphia, PA: USENIX Association.

Shah, S. C. (2017). Recent advances in mobile grid and cloud computing. Intelligent Automation & Soft Computing, 1-13.

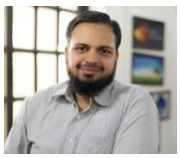
Singh, R., Sharma, U., Cecchet, E., & Shenoy, P. (2010). Autonomic mix-aware provisioning for non-stationary data center workloads. In *ICAC '10: Proceedings of the 7th IEEE International Conference on Autonomic Computing and Communication*. Washington, DC, USA: IEEE Computer Society.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., & Tantawi, A. (2005). An analytical model for multi-tier internet services and its applications. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (pp. 291–302). ACM volume 33.

Villela, D., Pradhan, P., & Rubenstein, D. (2007). Provisioning servers in the application tier for e-commerce systems. *ACM Transaction on Internet Technology, 7*.

Yazdanov, L., & Fetzer, C. (2014). Lightweight automatic resource scaling for multi-tier web applications. In *7th IEEE International Conference on Cloud Computing*. IEEE.

8 NOTES ON CONTRIBUTORS



Abid Nisar is the head of Software Development at Analytics, Private Limited, Lahore Pakistan. He has 12 years of experience building large scale software systems. Abid completed his MPhil in Computer Science from PUCIT, University of the Punjab, Lahore, Pakistan in 2015. His research interests are in cloud computing, machine learning, software architectures, and databases.

Waheed Iqbal is an assistant professor at Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan. He also worked as a Postdoc researcher with the Department of Computer Science and Engineering, Qatar University during 2017--2018. His research interests include cloud computing, distribute systems, machine learning, and large scale system performance evaluation. He received his Ph.D. degree from the Asian Institute of Technology, Thailand. He received dual Masters degrees in Computer Science and Information Technology from the Asian Institute of Technology and the Technical University of Catalonia (UPC), Barcelona, Spain, respectively.

Fawaz Bokhari is an assistant professor at the University of the Punjab - College of Information Technology (P.U.C.I.T). He is a Fulbright scholar and was awarded this scholarship in 2007 for his Ph.D. studies in United States. He received his Ph.D. in Computer Science from the University of Texas at Arlington in 2012. His research interests include network protocols design (Layer 2,3 & 4 of the TCP/IP stack) for wireless and wired networks, resource provisioning of IoT based applications in the cloud, and TCP and routing for datacenters.

Faisal Bukhari received M.Sc. in Statistics from the Institute of Statistics, University of the Punjab (PU), Lahore, Pakistan. He received M.Sc. in Computer Science from Punjab University College of Information Technology (PUCIT), PU. He also received a M.S. and Ph.D. in Computer Science from the Asian Institute of Technology (AIT), Thailand. Currently, he is an Assistant Professor and running the Imaging and Data Science Lab at PUCIT. His research interests include computer vision, image processing, data science, and machine learning.

Khaled Almustafa Received a B.E.Sc. in Electrical Engineering, M.E.Sc. and Ph.D. in Wireless Communication from the University of Western Ontario, London, Ontario, Canada in 2003, 2004 and 2007 respectively. He is currently working as an Associate Professor at Prince Sultan University (PSU) in the Department of Information Systems (IS) at the College of Computer Science and Information Systems (CCIS), Riyadh, K.S.A. He served as a General Supervisor for the Information Technology and Computer Services Center (ITCS) at PSU, Chairman of the Department of Communication and Networks Engineering (CME), and the Vice Dean for the College of Engineering at PSU. Currently he is the Director of the Research and Initiatives Center at PSU. His research interests include error performance evaluation of MIMO communication systems in partially known channels, adaptive modulation, and channel security, text recognition models, control systems with renewable energy applications as well as features selections and data pre-possessing.