Simultaneous Cost and QoS Optimization for Cloud Resource Allocation

Seyedehmehrnaz Mireslami, Student Member, IEEE, Logan Rakai, Member, IEEE, Behrouz Homayoun Far, Member, IEEE, and Mea Wang, Member, IEEE

Abstract—Cloud computing is a new era of computing that offers resources and services for Web applications. Selection of optimal cloud resources is the main goal in cloud resource allocation. Sometimes, customers pay more than required since cloud providers’ pricing strategy is designed for the interest of the providers. Nonetheless, cloud customers are interested in selecting cloud resources to meet their quality of service (QoS) requirements. Thus, for the interest of both providers and customers, it is vital to balance the two conflicting objectives of deployment cost and QoS performance. In this paper, we present a cost-effective and runtime friendly algorithm that minimizes the deployment cost while meeting the QoS performance requirements. In other words, the algorithm offers an optimal choice, from customers’ point of view, for deploying a Web application in cloud environment. The multi-objective optimization algorithm minimizes cost and maximizes QoS performance simultaneously. The proposed algorithm is verified by a series of experiments on different workload scenarios deployed in two distinct cloud providers. The results show that the proposed algorithm finds the optimal combination of cloud resources that provides a balanced trade-off between deployment cost and QoS performance in relatively low runtime.

Index Terms—Cloud computing, quality of service, multi-objective optimization, Web application deployment.

I. INTRODUCTION

CLOUD computing is known as a pool of cloud resources such as storage, database servers, and computing servers. Cloud providers deliver these resources to customers through the Internet. Enterprises such as Netflix [1] and Dropbox [2] choose cloud computing to deploy their services as a cost-effective solution. Another example of a large-scale cloud-based Web application is the popular online shopping website, ebay [3].

On one hand, the growth of cloud-based Web application deployment challenges cloud providers to efficiently utilize cloud resources. On the other hand, cloud customers’ expectation in terms of QoS continues to grow as cloud-based services are becoming more common. Resource allocation in cloud datacenter meeting the interests of both cloud providers and customers attracts attention in both research and development of cloud infrastructure. Recent works consider different strategies to solve the resource allocation problem [4]–[6]. Various characteristics of the resources such as on-demand self-service and broad network access offered by cloud providers, make the resource allocation problem challenging to the cloud customers. Cloud providers usually have complex pricing model and Quality of Service (QoS) for their resources. For example, Amazon Web Services [7] offers different types of services for computing, storage, database and networking with different prices. Therefore, it is challenging for customers to select an appropriate combination of resources within their budget considering QoS requirements [4], [8]–[10]. For example, different types of computing instances are offered by Amazon Web Services such as Amazon EC2, Amazon Lambda, Amazon EC2 Container Registry, etc. In addition, each of these instance types can have different characteristics, such as storage of each instance, memory and specific operating system, that affect the price.

There are several cloud management tools, such as RightScale [11], IBM SmartCloud [12] and AWS OpsWorks [13], that help customers manage their resources by scaling up or scaling down the instances allocated to a service in the cloud environment [7], [14]. Usually, scaling the resources is done based on the application workload. These tools try to minimize both or either of the deployment cost and workload demand but ignore the QoS performance that is mostly critical from the cloud customers’ perspective.

Therefore, several research works have focused on optimizing QoS performance as well as deployment costs [8], [15]. In [8], the objective is minimizing total deployment cost and latency is considered as QoS criteria. In addition, mathematical formulation is developed for instances response time in [15] to satisfy customers’ QoS requirements.

The resource allocation problem in cloud datacenter is commonly modeled as a single objective optimization problem [8], [15]–[18]. These models consider either the total deployment cost, QoS performance, or energy consumption as the optimization objective. In [8], [15], and [19], minimizing deployment cost is the main objective and QoS performance requirements are considered as problems’ constraints. In [16], the single objective optimization problem for resource allocation problem is defined by maximizing QoS performance as

Manuscript received September 12, 2016; revised April 21, 2017 and July 13, 2017; accepted July 28, 2017. Date of publication August 11, 2017; date of current version September 1, 2017. This research is supported by the Natural Sciences and Engineering Research Council of Canada and Alberta Innovates Technology Futures. The associate editor coordinating the review of this paper and approving it for publication was F. De Turck. (Corresponding author: Seyedehmehrnaz Mireslami.)

S. Mireslami, L. Rakai, and B. H. Far are with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB T2N1N4, Canada (e-mail: smiresla@ucalgary.ca; lmrakai@ucalgary.ca; far@ucalgary.ca).

M. Wang is with the Department of Computer Science, University of Calgary, Calgary, AB T2N1N4, Canada (e-mail: meawang@ucalgary.ca).

Digital Object Identifier 10.1109/TNSM.2017.2738026
problem objective and response time is used as QoS criteria. Moreover, the main focus of [17] and [18] is minimizing total energy consumption. Such single-objective models often over optimize the objective and lead to degradations in other factors. For example, in [18], reducing energy consumption is the main goal. Therefore, it over optimizes energy consumption by sacrificing the QoS performance. In this paper, we employ a multi-objective optimization approach to solve the cloud resource allocation problem. In particular, we seek to optimize the deployment cost and QoS performance simultaneously. Our proposed optimal cloud resource allocation algorithm is specifically designed for Web application deployment in cloud datacenter. The comprehensive deployment cost and QoS performance models are developed in this work by considering the distinction between database instances and computing instances.

Though the deployment cost and QoS performance are two conflicting objectives, they are both important for cloud providers and customers. Selecting a proper trade-off among the total deployment cost and QoS performance objectives depends on the application’s specifications and customer’s desires. Our optimization algorithm seeks a balanced trade-off between the two objectives. The proposed optimization algorithm adopts a self-tuning method proposed in [20] to automatically find a set of multi-objective weights that leads to balanced optimization of both deployment cost and QoS performance. The advantage of this algorithm is that it eliminates the need for any tuning from the customer and can handle the difference in the scales of the two design objectives (deployment cost and QoS performance), which is essential for dynamic resource allocation.

In order to evaluate our proposed optimal cloud resource allocation algorithm, Amazon Web Services (AWS) [7] and RackSpace [21], which are two well known large managed cloud providers, are used. The performance of our proposed optimal cloud resource allocation algorithm is evaluated by comparing its results with AWS OpsWorks, a single objective QoS-aware deployment cost optimization algorithm (QCost algorithm), and the conventional method which are presented in [15]. The conventional resource allocation method is the modified version of the QCost algorithm to consider different types of instances as general virtual instances. AWS OpsWorks is an application management service that helps customers to configure and operate their applications by automatically scaling the resources [13].

For the experiments, seven different scenarios of two benchmark workloads are used. The experiments demonstrate the effectiveness of the proposed optimal cloud resource allocation algorithm in improving both deployment cost and QoS performance compared to the QCost algorithm, AWS OpsWorks, and the conventional method [15]. In addition, we have tested several large-scale workload scenarios including an industrial testcase to show the scalability of our algorithm. The algorithm’s short runtime makes it a good choice for dynamic resource allocation tools.

The rest of this paper is organized as follows. We review related works towards single-objective and multi-objective resource optimization in cloud-based Web application deployment in Section II. We then define the preliminaries in Section III, and present the proposed optimal cloud resource allocation algorithm in Section IV. In Section V, we validate the algorithm with simulation results based on real price models and benchmark workloads. Section VI concludes the paper.

II. BACKGROUND

A. Multi-Objective Optimization

Multi-objective optimization problems involve several objectives that often conflict. The general form of a multi-objective optimization problem is expressed as:

\[
\min_{x} \ [h_1(x) \cdots h_N(x)]
\]

\[\text{s.t. } x \in X\]

where it is aimed to find a solution in the feasible set \(X\) that minimizes the objectives \(h_1(x), \ldots, h_N(x)\). Since these objective components often conflict, they cannot be minimized simultaneously. In other words, decreasing one of the objectives may increase at least one of the other ones. Thus, there exist several optimal solutions for a multi-objective problem. All these solutions are optimal but offer different trade-off among the objectives. These solutions are called Pareto optimal solutions that are the solutions that cannot be dominated by any other solution. This means that there is no solution that achieves lower values for all objectives compared to Pareto optimal solutions.

The set of all Pareto optimal solutions is called Pareto front that is also known as the optimal trade-off curve. One of the Pareto optimal solutions may be chosen depending on the priorities and preferences of the domain expert. However, in most applications, the extreme solutions that largely minimize one objective but significantly degrade the others are not desirable. Often, the solutions with balanced trade-off among the objectives are sought [22], [23].

B. Web Application Deployment in Cloud Environment

Cloud computing is a new paradigm that significantly affects the information technology practices. Cloud computing is considered as a pool of on-demand Internet-based computing resources for serving large-scale applications. The demands for cloud resources have been growing rapidly as large enterprises such as Netflix and Dropbox have adopted cloud solutions. Resource allocation is a critical subject that brings significant research efforts towards the cloud computing field [4], [9], [10], [24]–[30]. With growing competition among software enterprises, QoS performance requirements have become a vital issue that should be considered when allocating resources in the cloud to provide a service. In this paper, we focus on two conflicting objectives: deployment cost (the cost of the cloud resources allocated to a service) and the QoS performance (the Quality-of-Service provided by the allocated resources). Single objective or multi-objective optimization techniques are used to model and solve resource allocation problems in cloud computing environment.

There are several works that address deployment cost optimization in cloud datacenters [8], [9], [16], [25].
Towards minimizing deployment cost, Aniceto et al. [9], propose an algorithm to minimize deployment service cost in a cloud environment. In this algorithm, a single cloud provider is considered for both short term and long term deployment periods. The number of required reserved and on-demand instances are estimated. Then, these estimates are used to reduce the deployment cost. The results show that the deployment cost is reduced by 32% with performance degradation as little as 5%. However, the runtime may be long for certain workloads since the algorithm includes several repeating estimation steps that may be time consuming. To improve the efficiency of the algorithm for dynamic resource allocation, Hu et al. [4] proposed an online algorithm that minimizes cost and resource rate by modeling the problem as a 2-dimensional parking permit problem. A deterministic algorithm is used for solving this problem. The work assumed that cloud provider offers different contracts on resource rate and duration. Instead of buying different small contracts to meet the demand, it is proposed to buy the contracts that may have overlap with others in the time-demand plan. This way, the small contracts are removed to reduce the cost.

To improve QoS, Huang et al. [26], propose an algorithm for service provisioning in a virtualized cloud environment. In order to optimize the user’s experience, the proposed algorithm considers QoS requirements in service composition. The problem is modeled as a directed graph. The algorithm finds the shortest path in the graph as the resource allocation solution for the least runtime. Although the results of the proposed algorithm show improvements in performance and runtime results, it does not provide any formulation and definition for QoS requirements. To this end, in both [8] and [31], QoS requirements are defined based on Service Level Agreement (SLA) and Service Confidence Level (SCL). Thus, the resource allocation is solved from the cloud customer’s point of view rather than the cloud provider’s. The works formulate the problem as a multi-cloud resource allocation problem focusing on QoS requirements. The main objective is to minimize total deployment cost while QoS requirements are considered as problem constraints. Since customer can rent virtual appliances and virtual machines from different cloud providers, in [8], the cost of purchasing a server includes costs of virtual appliances and costs of virtual machines. This leads to significant saving in the total deployment costs. However, it is not mentioned how latency, SCL and SLA are measured for defining QoS criteria. In [31], a genetic algorithm is adopted to solve the cloud resource allocation problem and QoS requirements are clearly quantified. Nonetheless, the runtime of the algorithm for large services is huge due to the complexity of the genetic algorithm. In addition, this approach needs a pre-specified application configuration, i.e., number of servers and type of servers, as an input and only outputs the optimal cloud provider assignment, not the actual resource combination.

Considering both the deployment cost and QoS requirements, Goudarzi et al. [28] formulate the cloud resource allocation problem. To solve this problem, dynamic programming is used. Minimizing deployment cost is considered as the objective of the problem while customer requirements are included as problem constraints. To find a balanced trade-off between these two objectives, a penalty is defined for cloud service providers in case they fail to satisfy customers’ QoS requirements. In this model, response time and client-level service agreement are considered as customer requirements, though client-level service agreement is not quantified. Similarly, Mireslami et al. [15] proposed an optimization algorithm to minimize total cost of Web-application deployment while satisfying QoS requirements. In this work, QoS requirements are defined as the response time of different resource types, e.g., database and computing instances. This problem is modeled as a single objective optimization problem where total deployment cost is the problem objective. All QoS requirements are considered as problem constraints. The experimental results show that this algorithm achieves a low total deployment cost while all the customers’ QoS requirements are met.

Works reviewed so far model the problem as a single objective optimization problem by considering total deployment cost as a problem objective and QoS requirements as problem constraints. This way, total deployment cost may be over-optimized at the expense of QoS degradation. Unlike these existing works, in this paper, a multi-objective algorithm is proposed to optimize both deployment cost and QoS performance requirements. The solution of the proposed algorithm provides a balanced trade-off between total deployment cost and QoS performance. To this end, Feng et al. [29] proposed a multi-objective model considering the total task executing time, QoS requirements of each task, and resource reservation cost, up on which they proposed an efficient task scheduler to allocate cloud resources. To model the problem as an optimization problem, a new algorithm based on particle swarm optimization is proposed. An optimal solution is found using Pareto-dominate theory. Although it can find the optimal solution unlike greedy algorithms and genetic algorithms, it does not provide any description and formulation for QoS requirements. As the scheduling problem scales, the runtime significantly increases since the number of particles that should be considered for each task grows. Furthermore, this work only improves the efficiency by decreasing the total execution time and does not reduce the amount of required virtual resources.

Utilizing multi-objective optimization methodology, Malekloo and Kara [32] also proposed an approach to minimize energy communication cost, power consumption and resource wastage in a cloud computing environment. An ant colony algorithm is used to solve this problem. This work improves the resource utilization to reduce the energy consumption and decrease datacenter costs. Therefore, a virtual machine placement algorithm is developed to solve the NP-hard problem in a multi-objective optimization approach. However, the algorithm does not scale very well for large-scale problems.

In this paper, a multi-objective optimization approach is employed to solve the resource allocation problem for deploying large-scale Web applications in cloud environment. Unlike the existing works, in this paper, both total deployment cost and QoS performance requirements are optimized simultaneously. This way, we prevent over optimizing one of the
problem objectives at the cost of degradation in the other one. Moreover, in this paper, the distinction between database instances and computing instances is considered to provide comprehensive models for deployment cost and QoS performance requirements which are usually ignored when solving the resource allocation problem.

III. PRELIMINARIES

Cloud providers offer different types of resources with different prices for deploying applications in cloud environments. Selecting the appropriate combination of resources based on the expected user demands is a daunting task. In this section, we define the problem preliminaries by reviewing an existing single-objective formulation for the problem.

A. Problem Formulation

In order to formulate the optimal cloud resource allocation problem, the information of the resources that are offered by cloud providers are collected as the input to the problem. Although cloud providers offer similar resources to customers, they are different in terms of QoS performance, price range, and type of services. Cloud providers offer detailed information about the required customers’ resources. This input is presented in Table I. As shown in this table, a customer needs to pay a price for database instance (\(P_{db}\)), computing instance (\(P_{c}\)), I/O request capacity (\(P_{IO}\)), and storage space (\(P_{s}\)). In this paper, deployment in a single cloud provider is considered, hence, the in-bound and out-bound price between different geographical regions of the cloud provider is almost negligible. That is why the in-bound and out-bound communications are not differentiated. Also, the service rate of a computing instance (\(R_{c}\)) and a database instance (\(R_{db}\)) are introduced by cloud providers as part of service level agreements.

In addition, the problem also includes customer requirements, in this case the requirements of the Web application, such as budget and QoS performance constraints as input. In Table II, the requirements of the Web application (customer) are represented. The Web application total deployment service duration is shown with \(D\). Expected demand for computing instances (\(d_{c}\)) and database instances (\(d_{db}\)) are used to represent the Web application workload. The customer’s expected maximum response time for computing instances (timeLimit\(_{c}\)) and database instances (timeLimit\(_{db}\)) are considered as customers’ QoS performance requirements. The required storage space for the Web application is represented by \(S\). Also, the maximum budget that customer allocates for deploying a Web application is shown with \(B\).

Web applications have different levels of required resources and QoS performance. In order to satisfy the Web application requirements, enough resources should be purchased by the customers. The cloud resources which the customer will be charged for, are:

- \(X_{c}\): Number of computing instances.
- \(X_{db}\): Number of database instances.
- \(X_{s}\): Required storage for each database instance (GB).
- \(X_{Rc}\): Minimum service rate provided by computing instances (\(\frac{1}{\text{hour}}\)).
- \(X_{Rdb}\): Minimum service rate provided by database instances (\(\frac{1}{\text{hour}}\)).

The proposed algorithm is developed to find the most appropriate combination of these cloud resources. In other words, these are the cloud resource allocation problem variables that need to be optimally found by the proposed algorithm.

Deployment cost is defined based on the Web application required resources as:

\[
TC = X_{db} \times (P_{db} + P_{c} \times X_{c}) \times D + X_{s} \times P_{c} \times D + [X_{Rdb} + X_{Rc}] \times P_{IO} \times D
\]

(1)

In (1), for service time \(D\), the total deployment cost includes the number of database servers (\(X_{db}\)) which includes the cost of a database instance (\(P_{db}\)) and its required storage (\(P_{s} \times X_{s}\)). \(P_{c}\) is the cost of a computing instance. Customer also should pay for communication between computing and database servers (\(X_{Rdb} + X_{Rc}\)) \(\times P_{IO}\) where, \(P_{IO}\) is the cost of I/O request capacity and \((X_{Rdb} + X_{Rc})\) is the summation of database and computing instances’ service rates.

B. Single Objective QoS-Aware Cost Optimization

To model the resource allocation problem as a single objective optimization, the total deployment cost is often considered as the problem objective and all the customer’s QoS performance are considered as problem constraints. To achieve

\[
\text{TABLE I}
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Input Parameter (unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_{db})</td>
<td>Unit cost of database instance ($/hour)</td>
</tr>
<tr>
<td>(P_{c})</td>
<td>Unit cost of computing instance ($/hour)</td>
</tr>
<tr>
<td>(P_{s})</td>
<td>Unit cost of storage for each database instance ($/GB)</td>
</tr>
<tr>
<td>(P_{IO})</td>
<td>I/O cost of request capacity ($/hour)</td>
</tr>
<tr>
<td>(R_{c})</td>
<td>Computing service rate: # of requests per hour a computing instance can handle ((\frac{1}{\text{hour}}))</td>
</tr>
<tr>
<td>(R_{db})</td>
<td>Database service rate: # of requests per hour a database instance can handle ((\frac{1}{\text{hour}}))</td>
</tr>
</tbody>
</table>

\[
\text{TABLE II}
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Input Parameter (unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D)</td>
<td>Total deployment service duration of the web application (hour)</td>
</tr>
<tr>
<td>(d_{c})</td>
<td>Expected computing demand: # of user requests per hour for computing servers ((\frac{1}{\text{hour}}))</td>
</tr>
<tr>
<td>(d_{db})</td>
<td>Expected database demand: # of user requests per hour to access the database server ((\frac{1}{\text{hour}}))</td>
</tr>
<tr>
<td>(\text{timeLimit}_{db})</td>
<td>Database max response time: maximum time for accessing database server (hour)</td>
</tr>
<tr>
<td>(\text{timeLimit}_{c})</td>
<td>Computing instance max response time: maximum time for serving a user request (hour)</td>
</tr>
<tr>
<td>(S)</td>
<td>Required storage for web application (GB)</td>
</tr>
<tr>
<td>(B)</td>
<td>Maximum budget allocated for web application deployment ($)</td>
</tr>
</tbody>
</table>
the optimal solution for deploying cloud-based Web application, a QoS-aware cost optimization formulation (QCost) is proposed in [15] as:

\[ \min_{X_{db}, X_c, X_s, X_{Rdb}, X_{Rc}} \ TC \]

s.t.
1: \( TC \leq B \)
2: \( \text{time}_{db}(X_{Rdb}) \leq \text{timeLimit}_{db} \)
3: \( \text{time}_c(X_{Rc}) \leq \text{timeLimit}_c \)
4: \( X_c \times R_c \geq d_c \)
5: \( X_{db} \times R_{db} \geq d_{db} \)
6: \( 1 \leq X_c \)
7: \( 1 \leq X_{db} \)
8: \( X_{db} \times X_s \geq S \)
9: \( \min_s \leq X_s \leq \max_s \)
10: \( X_c \in \{1, 2, 3, \ldots\} \)
11: \( X_{db} \in \{1, 2, 3, \ldots\} \)

In Constraint 1, customer’s budget limits are considered. In Constraints 2 and 3, it is ensured that the response times of database and computing instances satisfy customers’ QoS performance requirements. Constraints 4-7 indicate that at least one database instance and one computing instance should be purchased and check that all the purchased instances can handle users’ demand. Also, the customer should at least purchase enough storage space for deploying the Web application that is enforced by Constraint 8. In addition, Constraint 9 represents that the purchased storage should be in the offered range by cloud provider, i.e., between minimum offered storage (min_s) and maximum offered storage (max_s). Finally, customers should purchase discrete number of instances that are considered by Constraints 10 and 11.

The QoS-aware cost optimization (QCost) formulation [15], presented in (2), achieves an appropriate combination of cloud resources that minimizes total deployment cost while QoS performance requirements are satisfied. Although this formulation can find a combination of cloud resources for deploying a Web application with low cost and acceptable QoS performance, it has several shortcomings which are explained in the followings:

- In this formulation, the QoS, i.e., database response time and computing response time constraints, requires the customer to carefully set the bounds. Often, setting the constraint bounds is a challenging task as it is time consuming and needs several experiments to find the desired trade-off between deployment cost and QoS performance.
- The focus of this QoS-aware cost optimization formulation is minimizing the total deployment costs. If the QoS constraints’ bounds are not set properly, it may lead to sacrificing QoS performance to optimize the total deployment costs resulting in an unbalanced solution.

In [15], the QCost formulation only minimizes total deployment costs while the QoS performance requirements such as database response time and computing response time are considered as problem constraints. In order to prevent over optimizing total deployment costs by largely degrading the QoS performance, a multi-objective optimal cloud resource allocation algorithm is proposed in Section IV to overcome the two shortcomings mentioned above. The proposed optimal cloud resource allocation algorithm can achieve a solution with a balanced trade-off among the total deployment costs and QoS performance with no significant runtime overhead.

IV. COST-EFFICIENT QoS-AWARE CLOUD RESOURCE ALLOCATION

The nature of many engineering problems consist of several conflicting factors that should be considered simultaneously. In this paper, our goal is to find a balance between deployment cost and QoS performance in cloud resource allocation when deploying a Web application. We employ multi-objective optimization to find the optimal solution by considering the trade-off between the two conflicting objectives: minimum deployment cost and maximum QoS performance guarantee. In this section, first, the multi-objective cloud resource allocation formulation is explained. Then, the optimal cloud resource allocation algorithm is proposed to achieve the best combination of cloud resources for Web application deployment.

A. Multi-Objective Cloud Resource Allocation Formulation

The two conflicting objectives of the cloud resource allocation problem are minimizing the total deployment cost and maximizing QoS performance. Considering the complex nature of QoS criteria and cloud provider pricing models, striking a balanced trade-off between them is challenging. In this paper, multi-objective optimization is applied to simultaneously minimize total deployment cost and maximize QoS performance by automatically finding a balanced trade-off. To achieve the problem formulation, we first define the QoS performance requirements as database and computing response times, and then develop the multi-objective optimization formulation.

1) Defining QoS Performance Requirements: To propose a customer-oriented Web application deployment model, several customer requirements need to be met. Such requirements are database response time and computing response time that are often considered as customer constraints [15], [33]. The database response time and computing response time models are developed in this paper to represent QoS performance requirements in the problem formulation.

The database response time significantly depends on the expected database demand (\( d_{db} \)). The demand for database is modeled using an arrival rate following a Poisson process where the mean service time is (1/\( X_{Rdb} \)) [33]. Thus, the database response time of a Web application with a database service rate of \( X_{Rdb} \) is modeled as:

\[ \text{time}_{db}(X_{Rdb}) = \frac{1}{X_{Rdb}} \]

(3)

The database response time should be less than timeLimit_{db} which is the maximum allowed response time for database instances specified by the customer:

\[ \text{time}_{db}(X_{Rdb}) \leq \text{timeLimit}_{db} \]

(4)
Response time of the computing instances for serving a user request is another major concern in Web application deployment problem. Suppose $X_{Rs}$ is the minimum service rate of all computing instances and actual arrival rate of user requests for computing instances is shown by $d_c$. Following the same procedure as derivation of the database response time, the computing response time model is proposed as:

$$timec(X_{Rs}) = \frac{1}{X_{Rs}} \times \frac{1}{1 - \frac{d_c}{X_{Rs}}} \quad (5)$$

Same as database response time, computing response time should be less than $timeLimit_c$, that is the maximum allowed response time for computing instances specified by the customer:

$$timec(X_{Rs}) \leq timeLimit_c. \quad (6)$$

2) Multi-Objective Optimization Formulation: As mentioned in Section III, the QoS-aware cost optimization formulation in (2) minimizes total deployment cost by considering QoS performance requirements as problem constraints. Therefore, QoS performance may be sacrificed due to over optimizing total deployment costs. To overcome this problem, a multi-objective approach is proposed in this paper. Total deployment costs and QoS performance requirements are considered as objective components. However, improving one may lead to degradation of the other one. The proposed multi-objective approach is capable of achieving a balanced trade-off between total deployment cost and QoS performance requirements where the values of both objectives are optimized as much as possible.

In common multi-objective techniques, a weighted sum of objective components with constant weights is considered as the target of minimization. Such weighted objective function for the multi-objective cloud resource allocation can be formulated as:

$$a_1 \times TC + a_2 \times (upperDBTime) + a_3 \times (upperCompTime) \quad (7)$$

where new variables $upperDBTime$ and $upperCompTime$ are defined as upper bound variables of response times for database and computing instances. $a_1$, $a_2$, and $a_3$ are non-negative and their sum is 1. This formulation has the same shortcoming as the formulation in (2). The constant multi-objective weights $a_1$, $a_2$, and $a_3$ must be carefully set by the customer with the domain knowledge. There may be large differences between the scale of the objective components that may lead to unbalanced optimization of the components. Therefore, the customer needs to scale/normalize the objectives before solving the optimization problem.

To deal with this issue, the idea proposed in [20] for geometric programming is employed which is called Self-Tuning Multi-Objective framework. To apply the framework, in the first step, the constant weights $a_1$, $a_2$, and $a_3$ are converted to the optimization variables $\alpha_1$, $\alpha_2$, and $\alpha_3$. Then, the objective changes to minimizing $\alpha_1 \times TC + \alpha_2 \times (upperDBTime) + \alpha_3 \times (upperCompTime)$. It should be mentioned that this objective can alternatively be interpreted as minimizing the geometric mean of the individual objective functions. The variables $\alpha_1$, $\alpha_2$, and $\alpha_3$ are optimally found by the optimization solver and the customer does not need to set them beforehand. The constraint $\alpha_1 + \alpha_2 + \alpha_3 = 1$ is added to account for the inherent trade-off among the competing objectives. This is because total deployment cost, database response time and computing response time (QoS performance requirements) are competing meaning that improving one may lead to a degradation in the other one. These weights need to be non-negative otherwise the problem becomes unbounded. Therefore, a constraint $(\alpha_1, \alpha_2, \alpha_3 \geq 0)$ is added to consider the non-negativity.

Applying this self-tuning multi-objective framework, the objective and all constraints remain in Geometric Programming (GP) form except for the new $\alpha_1 + \alpha_2 + \alpha_3 = 1$ constraint, which is not a legal GP equality constraint and does not fit in GP format. For this reason, Farshidi et al. [20] propose to replace this constraint with $\alpha_1 \times \alpha_2 \times \alpha_3 = 1$ which is in legal form and allowed in GP. Using this constraint, the problem is then completely in GP form and can be solved efficiently using convex optimization techniques where it is guaranteed that the achieved solution is the best feasible solution [34]. The non-negativity of variables is implicit in GP. In other words in GP, all the variables are assumed to be non-negative as solving a GP involves a log transform on the variables.

In this paper, a multi-objective formulation for optimal cloud resource allocation problem is proposed. This formulation that optimizes total deployment cost and QoS performance requirements simultaneously is presented as:

$$\min_{X_{db}, X_c, X_{Rs}, \alpha_1, \alpha_2, \alpha_3, upperDBTime, upperCompTime} \alpha_1 \times TC + \alpha_2 \times (upperDBTime) + \alpha_3 \times (upperCompTime)$$

s.t.

1: $TC \leq B$
2: $\frac{1}{X_{db}} \times \frac{1}{1 - \frac{d_c}{X_{Rs}}} \leq upperDBTime$
3: $\frac{1}{X_c} \times \frac{1}{1 - \frac{X_c}{X_{Rs}}} \leq upperCompTime$
4: $X_c \times R_c \geq d_c$
5: $X_{db} \times R_{db} \geq d_{db}$
6: $X_c + X_{db} \geq 1$
7: $X_{db} \times X_c \geq S$
8: $min_s \leq X_c \leq max_s$
9: $upperDBTime \leq timeLimit_{db}$
10: $upperCompTime \leq timeLimit_c$
11: $X_c \in \{0, 1, 2, \ldots\}$
12: $X_{db} \in \{0, 1, 2, \ldots\}$
13: $\alpha_1 \times \alpha_2 \times \alpha_3 = 1$
14: $\alpha_1, \alpha_2, \alpha_3, X_{Rdb}, X_{Rs} > 0$
15: $upperDBTime, upperCompTime > 0 \quad (8)$
As shown in the formulation in (8), the main objective components are total deployment cost and QoS performance. The first component of the objective includes \( TC \) which represents total deployment cost. The other objective component includes upper bounds for response times of database and computing instances (upperDBTime and upperCompTime) which are used to optimize QoS performance. With non-negative variable multi-objective weights \( \alpha_1, \alpha_2, \) and \( \alpha_3 \), we strike a balanced trade-off between the deployment cost and QoS performance objective components without requiring customer tuning or objective scaling.

Constraint 1 represents budget limits on the total deployment cost that should be kept under the maximum allowed budget. B. Constraints 2 and 3 are added to optimize the QoS performance requirements that are database and computing response times. In these two constraints, database and computing response times are set to be less than upper bound variables upperDBTime and upperCompTime, respectively. By putting these upper bound variables in problem objective, we optimize the QoS performance as well as total deployment cost rather than only meeting the customers’ QoS requirements. Constraints 4 and 5 make sure that numbers of purchased computing and database instances can handle the user demands, \( d_c \) and \( d_{db} \), Constraint 6 is added to obtain a physically meaningful solution, i.e., at least one database or computing instance should be purchased. Constraints 7 and 8 ensure the purchased storage space for Web application is sufficient and is between the defined minimum storage (\( \min_s \)) and maximum storage (\( \max_s \)) of the cloud provider. We add Constraints 9 and 10 to make sure that the customer’s QoS performance requirements for database instance response time (\( \text{timeLimit}_{db} \)) and computing instance response time (\( \text{timeLimit}_c \)) are satisfied. Constraints 11 and 12 are added due to the fact that the cloud customer can only purchase discrete number of database and computing instances. Constraint 13 is added to represent the inherent trade-off between the objective components. Finally, Constraints 14 and 15 note the non-negativity of multi-objective weights and other variables.

In (8), the objective function and all the constraints except Constraints 2, 3, 11 and 12 are in GP form. Constraint 2 and 3 can be rearranged to convert the problem in GP form. Also, Constraints 11 and 12 should be relaxed by considering \( X_c \) and \( X_{db} \) as continuous variables. Since non-negativity of the variables is implicit in GP, Constraints 14 and 15 are not stated but implicitly considered by the GP solver. Therefore, the multi-objective optimal resource allocation problem is converted to a geometric program as follows. This problem can be solved using convex geometric programming techniques with a scalable complexity [35].

\[
\begin{align*}
\min & \quad \alpha_1 \times TC + \alpha_2 \times (\text{upperDBTime}) \\
& \quad + \alpha_3 \times (\text{upperCompTime}) \\
\text{s.t.} & \quad TC \times B^{-1} \leq 1
\end{align*}
\]

Algorithm 1 Optimal Cloud Resource Allocation Algorithm

**Input:** Parameters from cloud provider and web application

**Output:** Best Discrete Solution, \( X^* = [X_{db}^*, X_c^*] \)

1: \( \text{branches} = [] \) // no branch constraints to begin
2: // initially branch on \( X_{db} \)
3: \( \{X^*, \text{obj}^*\} = \text{BRANCH}\text{SOLVE}(\text{branches}, \text{db}) \)

\[
X_{Rdb}^{-1} \times (d_{db} + \text{upperDBTime}^{-1}) \leq 1
\]

\[
X_{Rc}^{-1} \times (d_c + \text{upperCompTime}^{-1}) \leq 1
\]

\[
X_c^{-1} \times R_c^{-1} \times d_c \leq 1
\]

\[
X_{db}^{-1} \times R_{db}^{-1} \times d_{db} \leq 1
\]

\[
(X_c + X_{db})^{-1} \leq 1
\]

\[
X_{db}^{-1} \times X_s^{-1} \times S \leq 1
\]

\[
\min \times X_c^{-1} \leq 1
\]

\[
X_c \times \max_s^{-1} \leq 1
\]

\[
\text{upperDBTime} \times \text{timeLimit}_{db}^{-1} \leq 1
\]

\[
\text{upperCompTime} \times \text{timeLimit}_c^{-1} \leq 1
\]

\[
\alpha_1 \times \alpha_2 \times \alpha_3 = 1
\]  (9)

The proposed multi-objective formulation has three main advantages:

- It eliminates the need for customer tuning on multi-objective variable weights. This is because the multi-objective variable weights \( \alpha_1, \alpha_2, \) and \( \alpha_3 \) are set automatically during the optimization procedure and there is no need for time consuming experiments to find the most appropriate multi-objective weights.
- It eliminates the need for objective scaling. In conventional multi-objective optimization, objectives need to be normalized or scaled in order to have comparable contributions in the weighted objective. In this new formulation, objective scales are balanced automatically since the multi-objective weights are considered as optimization variables. Therefore, the contributions of different objective components will be similar.
- The achieved solution provides a balanced trade-off between total deployment cost and QoS performance. This is experimentally demonstrated in Section V.

B. Optimal Cloud Resource Allocation Algorithm

The results of the proposed multi-objective cloud resource allocation formulation are guaranteed to be the global optimum (the best feasible solution) since it is a geometric program. However, the results of a geometric program are continuous. In reality, some cloud providers only offer discrete number of database instances and computing instances. To make the optimal solution feasible in reality, a global algorithm is proposed in Algorithm 1 to find the optimal cloud resources. This algorithm is developed based on the Branch-and-Bound (BB) [36] technique in order to output realistic discrete solutions.

The proposed optimal cloud resource allocation algorithm can be modeled and explained as a binary tree that is shown
in Fig. 1. The root of the tree represents the continuous multi-objective formulation of (9), $X_{db}^P$ is the solution from solving the multi-objective formulation at node $P_0$. Each node in the tree has a lower bound and an upper bound marking the continuous region represented by the node. For instance, in the first level of the tree, the lower bound constraint is added as $X_{db} \leq \lfloor X_{db}^P \rfloor$ on one branch and the upper bound constraint is added as $X_{db} > \lceil X_{db}^P \rceil$ on the other branch. In other words, at each node, a new constraint is added to the multi-objective resource allocation formulation and then, the multi-objective formulation is solved.

At each level of the tree, the lower bound and the upper bound constraints for one variable are added. The branching continues on $X_{db}$ until the discrete solution for that variable is reached. Then, in the next level as illustrated in Fig. 1, the tree continues with variable $X_c$ by adding the upper/lower bound constraint and solving the multi-objective formulation. For each node of the tree, the feasibility conditions should be checked. If the problem is infeasible, the tree stops branching at the corresponding node. In addition, the branching stops once the multi-objective formulation solutions for both database and computing instances are discrete. Finally, the proposed algorithm outputs the final solution that achieves the minimum objective value among the nodes with discrete solutions, i.e., the globally optimal discrete solution.

The proposed optimal cloud resource allocation algorithm is presented in detail in Algorithm 1 and Algorithm 2. According to Algorithm 1, the parameters from cloud provider and Web application are considered as inputs to the algorithm. The algorithm starts branching on variable $X_{db}$ by calling procedure $\text{BranchSolve}()$ presented in Algorithm 2. This procedure recursively solves the multi-objective formulation until it finds a discrete solution to the problem or the problem becomes infeasible at all branches. In line 1 of Algorithm 2, $\text{SOLVE}()$ function is called that runs the optimizer on one node in the tree. In this paper, Mosek 6.0 solver [37] is used as the optimizer to solve the multi-objective formulation. However, any optimization solver that can solve geometric programming can be used. The union operator means appending the constraints in branches to formulation of (9). The branching continues on variable $X_{db}$ until the discrete solution is reached. If the solution for $X_{db}$ is discrete, we fix its value by adding an equality constraint (line 7). Then, in line 8, we start branching on $X_c$ by adding upper or lower bound constraints to the formulation of (9) and solving the optimization problem. The branching on $X_c$ continues until the discrete solution for $X_c$ is achieved. If the problem becomes infeasible, Algorithm 2 returns an empty solution with infinity as the objective value to Algorithm 1 (line 2). Otherwise, the Algorithm 2 returns the solution $X$ and the objective value $obj$ to Algorithm 1 if discrete solutions for both $X_c$ and $X_{db}$ are achieved (line 5). This algorithm is based on branch and bound technique and takes a global optimization approach. Therefore, among all visited solutions, the algorithm choses the discrete solution that has the minimum objective value. This solution is globally optimal meaning that it is the best feasible discrete solution to the cloud resource allocation problem.

V. PERFORMANCE ANALYSIS

In this section, we present the performance analysis of our proposed optimal cloud resource allocation algorithm. We evaluate the algorithm using a set of deployment experiments.

A. Experimental Setup

The algorithm is developed in C++ environment that is run on an OS X El Capitan machine with 1.6 GHz Intel Core i5 and 8 GB of memory. GPs are solved using Mosek 6.0 [37]. The required data is gathered from two different workload benchmarks. For the first workload, RUBiS, the behaviors of an auction Web site users like eBay are simulated using a simulator [38]. The second workload is a multi-tier SPECjAppServer2004 that employs a client simulator to produce benchmarks [38]. The detailed characteristics and QoS performance from running these two workloads for 15 minutes are shown in Table III.
Seven different scenarios of RUBiS and SPEC workload benchmarks are presented in the first column of Table III. For each scenario, the maximum allowed response times of database and computing instances are presented in columns 2 and 3. In column 4, the required storage for each workload scenario is shown. Columns 5 and 6 present the expected demands for accessing database instances and computing instances.

In this paper, the well known Amazon Web Services (AWS) [7] and Rackspace [21] are considered as cloud providers. The required parameters from AWS and Rackspace for deploying Web application with the RUBiS and SPEC workloads are shown in Table IV.

In Table IV, the hourly price of purchasing one database instance and one computing instance are shown in columns 2 and 3. For each database instance, the hourly cost of storage per GB is presented in column 4. Cloud providers also charge customers for communication between database instance and computing instance that is shown in column 5. In columns 6 and 7, the service rates that each database and computing instance can handle are shown.

The results of the proposed optimal cloud resource allocation algorithm and the QCost algorithm proposed in [15] for deployment in AWS cloud environment are compared in Tables V and VI. These workload scenarios are deployed on two different cloud providers which are AWS and Rackspace cloud environments, respectively. In these tables, column 2 presents the first major objective of cloud resource allocation, i.e., total deployment cost ($TC$). In columns 3 and 4, the other objectives of cloud resource allocation, database response time ($time_{db}$) and computing response time ($time_{c}$), are shown respectively. In columns 5 to 7, the same results for the proposed algorithm are given.

The results represent that our proposed optimal cloud resource allocation algorithm outperforms the QCost algorithm. When considering the total deployment cost of the two algorithms, we observe that the cost achieved by the proposed algorithm is either better or equal to the total deployment cost achieved by the QCost algorithm except for one workload where a slight increase is seen. However, the superiority of the proposed algorithm is realized when comparing the QoS performance requirements of database and computing response times. Both QoS performance requirements achieved by the proposed optimal cloud resource allocation algorithm are largely lower than the ones from the QCost algorithm. This is actually expected because the QCost algorithm uses a single objective optimization approach where deployment cost is the main objective. However, our proposed algorithm optimizes both QoS performance and total deployment cost simultaneously that leads to large QoS improvements with no need to sacrifice the total deployment cost objective.

The results from the two tables indicate that our proposed optimal cloud resource allocation algorithm is not limited to a certain cloud provider and can be applied to optimally deploy Web applications into any cloud provider while offering a balanced trade-off among the QoS performance and total deployment cost objectives.

As an example, considering the Amazon Web Services (AWS) cloud provider, the largest improvement is achieved for workload scenario of RUBiS800, where the proposed optimal cloud resource allocation algorithm reduces the total deployment cost by about 51% compared to the QCost algorithm. Furthermore, the proposed optimal cloud resource allocation algorithm reduces the database response time by 83% and computing response time by 75% compared to the QCost algorithm.

The largest improvement for Rackspace cloud environment is also obtained for workload scenario of RUBiS800. Compared to the QCost algorithm, the proposed optimal cloud resource allocation algorithm reduces the total deployment cost to about 72%. Also, the database response time and
computing response are reduced to about 17% and 25%, respectively.

B. Trade-Off Analysis

The trade-off curves for Rubis3200 benchmark for database response time vs total deployment cost as well as computing response time vs total deployment cost are plotted in Fig. 2 and Fig. 3, respectively. The AWS is considered as the cloud provider in this experiment. The trade-off curves are obtained by considering the multi-objective weights of (9) as constants, setting different values for them and solving the optimization problem for each value. These curves show that the proposed optimal cloud resource allocation algorithm significantly outperforms the QCost algorithm. These figures also represent the shortcomings of QCost compared to the proposed optimal cloud resource allocation algorithm.

The main focus of QCost is finding the solution that only minimizes total deployment cost while meeting all QoS requirements. Therefore, as soon as QCost finds a solution with the lowest total deployment cost that satisfies the customers’ QoS requirements, it terminates. This way, the only goal of the optimizer is to meet the response time constraints rather than optimizing them. For example, as shown in Fig. 2, the black square is the solution of QCost algorithm. Although QCost algorithm minimizes the total deployment cost, it only satisfies the database response time constraints and does not try to optimize them. Our proposed optimal cloud resource allocation algorithm however finds the best possible solution considering both deployment cost and QoS performance, which is shown with a red circle. In other words, the proposed algorithm not only looks at the QoS performance but also tries to achieve a better solution by considering the QoS performance as a problem objective component rather than just as constraints. Customers often appreciate significant improvements in QoS performance at a small increase in cost. Therefore, it finds a balanced solution within the overall customers' requirements.

C. Proposed Optimal Cloud Resource Allocation Algorithm Vs Exhaustive Search Algorithm

In this paper, an exhaustive search algorithm is developed to help better show the effectiveness of proposed optimal cloud resource allocation algorithm. The exhaustive search algorithm considers all possible discrete solutions for both database instances and computing instances to find the best trade-off between total deployment cost and QoS performance requirements. In Table VII, the runtime results for both proposed optimal cloud resource allocation algorithm and the exhaustive search algorithm for deployment in both Amazon Web Services (AWS) and Rackspace cloud environments are compared.

In this table, in columns 2 and 3, the runtime results of the proposed optimal cloud resource allocation algorithm and the exhaustive search algorithm when AWS is considered as the cloud provider are shown. In columns 4 and 5, the same runtime results when deploying in Rackspace are given. The runtime results show that the proposed optimal cloud resource allocation algorithm achieves large speedups compared to a brute-force algorithm such as the exhaustive search algorithm. However, it still achieves the same global optimal solution as the exhaustive search algorithm.

D. Objective Scale Invariance

The proposed multi-objective cloud resource allocation formulation of (9) is independent of objective scaling and it does not need any customer tuning. For example, if the unit of response time is changed from millisecond to microsecond, the solution and the values of the other objectives remain unchanged. This means that in contrast to the common multi-objective methods, the proposed approach does not need any objective scaling since the scales will be automatically adjusted using the variable weights. Therefore, all the objective components have similar contributions in the optimization objective. In order to show that the proposed formulation is objective scale invariant, the scale of one objective component
TABLE VIII
DEMONSTRATION OF OBJECTIVE SCALE INVARIANCE PROPERTY OF THE PROPOSED
MULTI-OBJECTIVE CLOUD RESOURCE ALLOCATION FORMULATION

<table>
<thead>
<tr>
<th>Response Time Unit</th>
<th>Obj. 1 (Total Cost) ($)</th>
<th>Obj. 2 (upperDBTime)</th>
<th>Obj. 2 (upperCompTime)</th>
<th>α₁</th>
<th>α₂</th>
<th>α₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>second</td>
<td>0.4</td>
<td>1.00 × 10⁹</td>
<td>1.66 × 10⁹</td>
<td>2.14</td>
<td>0.528</td>
<td>0.881</td>
</tr>
<tr>
<td>millisecond</td>
<td>0.4</td>
<td>1.00 × 10⁶</td>
<td>1.66 × 10⁴</td>
<td>214.40</td>
<td>0.053</td>
<td>0.081</td>
</tr>
<tr>
<td>microsecond</td>
<td>0.4</td>
<td>1.00 × 10⁶</td>
<td>1.66 × 10⁴</td>
<td>21502.49</td>
<td>0.005</td>
<td>0.008</td>
</tr>
</tbody>
</table>

is changed to monitor the changes in the weights and values of the other objective component.

In this experiment, the unit of response time is changed from second to millisecond, and microsecond. As shown in Table VIII, the scales of objective component 2 (upperDBTime) and objective component 3 (upperCompTime) are changed. Then, the proposed multi-objective cloud resource allocation formulation is solved for each of these scales. In columns 2 to 4, the optimal values for objective component 1 (Total Cost), objective component 2 (upperDBTime) and objective component 3 (upperCompTime) are presented, respectively. In addition, the weights associated with these objectives $\alpha_1$, $\alpha_2$, and $\alpha_3$ are given in columns 5 to 7, respectively. As shown in Table VIII, the value of objective component 1 (Total Cost) remains unchanged, although the scale of the other two objective components are changed. These results demonstrate that the proposed multi-objective cloud resource allocation formulation does not need any objective scaling and the solution is independent of objective scales.

E. Performance Comparison

In order to validate the efficiency of the proposed optimal cloud resource allocation algorithm, in this experiment, the solutions of our proposed algorithm are compared with the solutions of QCost algorithm [15], AWS OpsWorks [13], and the conventional method [15]. Amazon Web Services (AWS) offers application management service which is called AWS OpsWorks to help customers in deploying and managing their applications in Amazon cloud environment. AWS OpsWorks scales applications according to user demands. The conventional method is presented in [15] which is the modified version of the QCost algorithm to consider different types of instances the same, as general virtual instances. Since in this paper, different types of instances are considered to model the proposed optimal cloud resource allocation algorithm, our results are compared to the conventional method to see how distinction among instances affects the results. As mentioned in Section III, QCost algorithm minimizes total deployment cost while QoS performance requirements are considered as problem constraints. The total deployment cost, database response time and computing response time are compared for these four methods in Fig. 4.

Fig. 4(a) and Fig. 4(b), compare database response time and computing response time of the proposed optimal cloud resource allocation algorithm, QCost algorithm, AWS OpsWorks, and the conventional method. In addition, the total deployment cost is compared for these methods in Fig. 4(c).

According to these figures, in terms of database and computing response times, the proposed cloud resource allocation algorithm significantly outperforms QCost algorithm, AWS.
OpsWorks, and conventional method. Although the proposed cloud resource allocation algorithm improves response times, it offers a slightly more expensive solution compared to the QCost algorithm and AWS OpsWorks. These figures correctly demonstrate that the proposed algorithm finds a balanced trade-off between QoS performance and deployment cost. However, even though the results show lower deployment costs using AWS OpsWorks, in fact, it will require a much more expensive solution to have QoS performance as good as the proposed algorithm’s solution.

F. Scalability Analysis

Due to the dynamic nature of the user demands, the turnaround time of any cloud resource allocation algorithm is a major concern. On the other hand, the algorithm should maintain a low runtime even when the Web applications scale. To provide a scalability analysis of our proposed algorithm, several experiments are performed in this section. We have verified the proposed algorithm by experimenting with large-scale workload scenarios. These workload scenarios are presented in Table IX.

The large-scale workload scenarios in this section include two distinct testcases, Testcase 1 and Testcase 2, presented in [40]. For these testcases, demands are obtained from Google Cluster Trace files. In addition, an industrial workload scenario from PYXIS WorldView [39] is used to provide a real word experiment. In order to better show the scalability of the proposed algorithm, in addition to these three workload scenarios, the scaled version of workload scenarios used in Section V-A are evaluated. These can provide a referenced comparison. The input parameters of these large-scale workload scenarios are demonstrated in Table IX. As shown in this table, for RUBiS and SPEC workload scenarios, only the user demands are scaled but the QoS are maintained the same as in Section V-A.

Table X includes the total cost, QoS and runtime results of our proposed algorithm for the large-scale workload scenarios. According to this table, for all experiments, the QoS requirements are satisfied. The total costs of the scaled scenarios has significantly increased as expected because more instances are now required to handle the large-scale user demands.

To visualize the impact of the workload scenario size on the proposed algorithm’s runtime, Fig. 5 is presented. In this figure, we compare the algorithm runtime for each original and scaled scenario. As expected, with a larger demand, an increasing trend in the runtime is seen. However, this increase is not proportional to the increase in the user demands. The most significant runtime increase is seen for SPEC40 workload scenario where the runtime increases from 0.08s to 0.15s (almost doubled) while the user demand size has increased by 20X. This demonstrates that the proposed algorithm scales well with the increases in user demands.

It should be mentioned that the size of the optimization problem, i.e., number of variables and number of constraints, does not change for larger scale scenarios. This is because in (9), we do not introduce one variable for each instance but we only have variables for the number of database and computing instances. However, the observed small increase in the runtime is expected as the solution search space for branch and bound may expand as the scenarios scale. Furthermore, the runtime of solving a GP problem not only depends on the number of variables and constraints, but is affected by sparsity and the tightness of the constraints.

VI. CONCLUSION

One of the challenging tasks in cloud resource allocation in datacenter is selecting an optimal combination of cloud

<table>
<thead>
<tr>
<th>Scenario</th>
<th>TimeLimit4</th>
<th>TimeLimit6</th>
<th>S (GB)</th>
<th>dmax</th>
<th>dc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testcase 1</td>
<td>0.65</td>
<td>0.31</td>
<td>420</td>
<td>14890</td>
<td>11940</td>
</tr>
<tr>
<td>Testcase 2</td>
<td>20.10</td>
<td>17.80</td>
<td>1920</td>
<td>2740</td>
<td>2740</td>
</tr>
<tr>
<td>RUBiS 800</td>
<td>10.00</td>
<td>4.00</td>
<td>320</td>
<td>32620</td>
<td>7060</td>
</tr>
<tr>
<td>RUBiS 1600</td>
<td>13.30</td>
<td>8.30</td>
<td>320</td>
<td>37040</td>
<td>25360</td>
</tr>
<tr>
<td>RUBiS 2400</td>
<td>14.60</td>
<td>10.50</td>
<td>320</td>
<td>40380</td>
<td>63920</td>
</tr>
<tr>
<td>RUBiS 3200</td>
<td>16.70</td>
<td>15.40</td>
<td>320</td>
<td>42940</td>
<td>202920</td>
</tr>
<tr>
<td>SPEC 10</td>
<td>9.20</td>
<td>11.70</td>
<td>760</td>
<td>69540</td>
<td>34680</td>
</tr>
<tr>
<td>SPEC 20</td>
<td>17.20</td>
<td>13.80</td>
<td>760</td>
<td>71260</td>
<td>37180</td>
</tr>
<tr>
<td>SPEC 40</td>
<td>19.80</td>
<td>17.50</td>
<td>760</td>
<td>73820</td>
<td>37260</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
<th>TC ($)</th>
<th>tmax</th>
<th>tmax</th>
<th>tmax</th>
<th>RunTime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYXIS WorldView</td>
<td>2.36</td>
<td>0.10</td>
<td>0.07</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>Testcase 1</td>
<td>5.61</td>
<td>3.30</td>
<td>4.37</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>Testcase 2</td>
<td>1.37</td>
<td>2.87</td>
<td>3.45</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>SPEC 10</td>
<td>7.81</td>
<td>1.66</td>
<td>1.00</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>SPEC 20</td>
<td>22.17</td>
<td>2.43</td>
<td>2.63</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>SPEC 40</td>
<td>29.61</td>
<td>2.78</td>
<td>3.45</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>SPEC 10</td>
<td>13.85</td>
<td>1.53</td>
<td>2.93</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>SPEC 20</td>
<td>26.86</td>
<td>2.87</td>
<td>3.45</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>SPEC 40</td>
<td>55.28</td>
<td>3.30</td>
<td>4.37</td>
<td>0.15</td>
<td></td>
</tr>
</tbody>
</table>
resources that meets the desired Quality of Service (QoS) performance with a relatively low cost. Cloud customers should go through the offered specifications from different providers to choose the services within their budget range. Selecting such combination of cloud resources is challenging due to the complex pricing methods that different cloud providers have. Unlike the existing works, in this paper, a multi-objective approach is taken to optimize total deployment cost and QoS performance at the same time from the cloud customers’ perspective. Then, an optimal cloud resource allocation algorithm based on branch and bound technique is proposed to obtain the optimal combination of cloud resources for meeting the customer’s requirements. A series of experiments on several Web applications with different workload scenarios and two well-known cloud providers is conducted in this paper to validate the effectiveness of the proposed algorithm. The results verify that an optimal combination of cloud resources is selected by the proposed optimal cloud resource allocation algorithm which achieves a balanced trade-off between deployment cost and QoS performance in a low runtime.

The future work may be extending the work by considering cloud resources from multiple cloud providers to obtain minimum deployment costs. Moreover, other types of applications such as multi-media, may be considered as a future direction of this research.

REFERENCES
Seyedehmehrnaaz Mireslami received the B.Sc. degree in applied mathematics from Tarfresh University, Iran, in 2011 and the M.Sc. degree in electrical engineering (software specialization) from the University of Calgary, Canada, in 2013, where she is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. Her research interests include cloud computing, quality of services, distributed systems, networking systems and algorithms, and optimization theory.

Logan Rakai (M’12) received the B.Sc. (Hons.) degree in computer engineering and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Calgary, Calgary, AB, Canada, in 2007, 2008, and 2012, respectively, where he is currently an Adjunct Professor with the Department of Electrical and Computer Engineering, Schulich School of Engineering and he strives to bring out the best in students.

He performed his industrial training with Mentor Graphics, Fremont, CA, USA. His current research interests include using distributed and high performance computing, machine learning, and specialized solvers to solve engineering problems in cloud computing, computer-aided design, and power systems.

Behrouz Homayoun Far has been a Professor of software engineering with the Schulich School of Engineering, University of Calgary, since 2001. His research interest areas are engineering of intelligent, distributed and heterogeneous networked systems, specifically on design, implementation, testing and verification of agent-based software systems. He established Intelligent Software Systems Laboratory, Schulich School of Engineering. He is the Co-Director of the Intelligent Transportation Laboratory, University of Calgary.

Mea Wang received the Bachelor of Computer Science (Hons.) degree from the Department of Computer Science, University of Manitoba, Canada, in 2002 and the Master of Applied Science and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Canada, in 2004 and 2008, respectively. She is currently an Associate Professor with the Department of Computer Science, University of Calgary. Her research interests include peer-to-peer networking, multimedia networking, cloud computing, as well as networking system design and development.