Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Using genetic algorithms to find optimal solution in a search space for a cloud predictive cost-driven decision maker

Ali Yadav Nikravesh, Samuel A. Ajila[*] and Chung-Horng Lung

Abstract

In a cloud computing environment there are two types of cost associated with the auto-scaling systems: resource cost and Service Level Agreement (SLA) violation cost. The goal of an auto-scaling system is to find a balance between these costs and minimize the total auto-scaling cost. However, the existing auto-scaling systems neglect the cloud client's cost preferences in minimizing the total auto-scaling cost. This paper presents a cost-driven decision maker which considers the cloud client's cost preferences and uses the genetic algorithm to configure a rule-based system to minimize the total auto-scaling cost. The proposed cost-driven decision maker together with a prediction suite makes a predictive auto-scaling system which is up to 25% more accurate than the Amazon auto-scaling system. The proposed auto-scaling system is scoped to the business tier of the cloud services. Furthermore, a simulation package is built to simulate the effect of VM boot-up time, Smart Kill, and configuration parameters on the cost factors of a rule-based decision maker.

Keywords: Self-adaptive auto-scaling systems, Cloud resource provisioning, Genetic algorithm, Cloud cost-driven decision maker, Virtual machine (VM), Service level agreement (SLA)

Introduction

The elastic nature of cloud computing enables cloud clients to benefit from the cloud's pay-as-you-go pricing model, which reduces cloud clients' capital expenses and their overall operational costs. However, maintaining Service Level Agreements (SLAs) with the end users obliges the cloud service provider to provide a certain level of Quality-of-Service (QoS) and the cloud service provider gets penalized if the cloud service fails to meet the desired SLAs.

Deciding the optimal amount of resources in a cloud computing environment is a double-edged sword which may lead to either under-provisioning or over-provisioning conditions. Under-provisioning condition is a result of saturation of the resources and may cause SLA violation. In contrast, over-provisioning condition occurs when the provisioned resources are wasted which results in excessive energy consumption and high operational cost [1]. Auto-scaling systems are developed to automatically balance a cost/performance trade-off and prevent the under-provisioning and over-provisioning conditions.

Figure 1 illustrates the typical stakeholders and their relationships in an Infrastructure-as-a-Service (IaaS) environment.

The three stakeholders in the IaaS environment are [2]:

- *Cloud infrastructure provider:* refers to the IaaS provider who offers logically unlimited virtual resources in the form of virtual machines (VMs), virtual networks, etc.
- *Cloud client:* is the customer of the IaaS provider who uses the infrastructure for hosting the cloud service. The cloud client also is known as the cloud service provider.
- *End user:* is the user that accesses the cloud service and generates the workload that drives the cloud service's behavior.

There are two types of SLAs in a cloud computing environment: SLAs between the end user and the cloud client, and SLAs between the cloud client and the cloud

* Correspondence: ajila@sce.carleton.ca
Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada

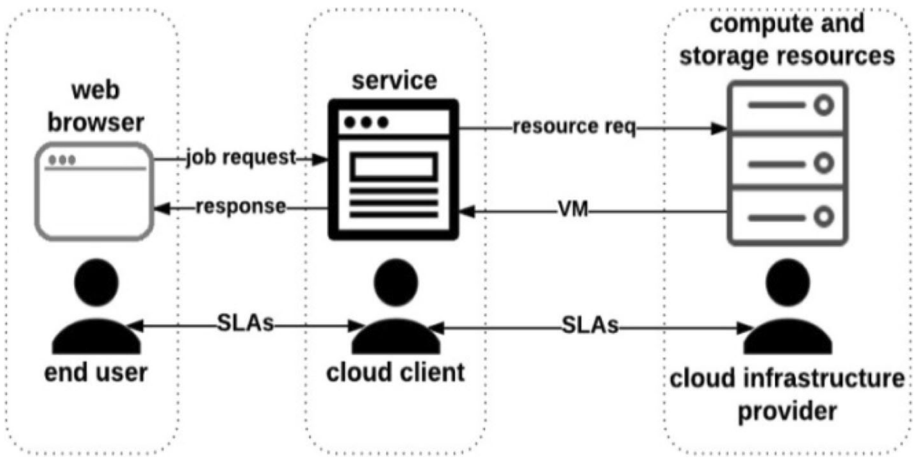

Fig. 1 Architecture of IaaS environment

infrastructure provider. This paper investigates the cost/performance trade-off from the cloud clients' perspective. From the cloud client's point-of-view the auto-scaling goal is to reduce resource cost (i.e., the cost of the leased resources from the IaaS provider) and the SLA violation cost (i.e., the cost that is associated with the SLA breaches), at the same time.

According to [3], rule-based systems are the most popular auto-scaling system in the commercial cloud computing environments. The rule-based systems reactively provision resources for the cloud service based on a set of scaling rules. However, the rule-based systems suffer from two main shortcomings [3]: a) their reactive nature, and b) the difficulty of selecting a correct set of configuration parameters. This paper investigates the impacts of these shortcomings on the accuracy of the rule-based systems and proposes an auto-scaling system to overcome the issues.

The reactive nature of the rule-based systems allows them to scaled-in or scaled-out a cloud service as soon as the performance of the cloud service reaches a predefined threshold. However, it takes between 5 and 15 min to boot-up a new VM and scaled-out the cloud service [4–6]. During the VM boot-up time the cloud service will be in the under-provisioning condition which may cause SLA violations. Therefore, the main shortcoming of the reactive auto-scaling systems (including the rule-based systems) is neglecting the VM boot-up time. The proposed auto-scaling system forecasts the future workload of the cloud service and generates the scaling requests ahead of time. This way, a new VM will be ready before the workload surge arrives to the cloud service.

The second shortcoming of using the rule-based systems is the configuration difficulty. A rule-based auto-scaling system has a set of configuration parameters which impacts its accuracy. Therefore, selecting the correct values for the configuration parameters is crucial in achieving an accurate auto-scaling system. In addition, the configuration values affect the auto-scaling system's decisions on how to balance the resource cost and the SLA violation cost. Since different cloud clients have different cost preferences, the auto-scaling system should be able to find a balance between the resource cost and the SLA violation cost based on the cloud clients' preferences. The proposed auto-scaling system uses genetic algorithm principle to automatically identify an optimum configuration of the rule-based systems. The focus of this paper is on the configuration issue. The proposed genetic algorithm considers the cloud client's cost preferences to find the optimum configuration set. Figure 2 shows the architecture of the proposed auto-scaling system and it consists of a *"self-adaptive prediction suite"* and a *"cost driven decision maker"*.

In our previous work [6] we proposed a self-adaptive prediction suite which automatically chooses the most suitable prediction algorithm based on the incoming workload pattern to forecast the future workload of the cloud service. In this paper we propose a cost-driven decision maker that minimizes the auto-scaling cost according to the cloud client's cost preferences. The research question here is: *"How to configure a rule-based decision maker to minimize the total auto-scaling cost based on the cloud clients' cost preferences?"*

The main contributions of this paper are:

- A novel cost driven decision maker to reduce the total auto-scaling cost based on the cloud client's preferences.
- An evaluation of our predictive auto-scaling system [6] against the Amazon auto-scaling system.
- An investigation of the impact of the VM boot-up time on the accuracy of the rule-based auto-scaling systems.

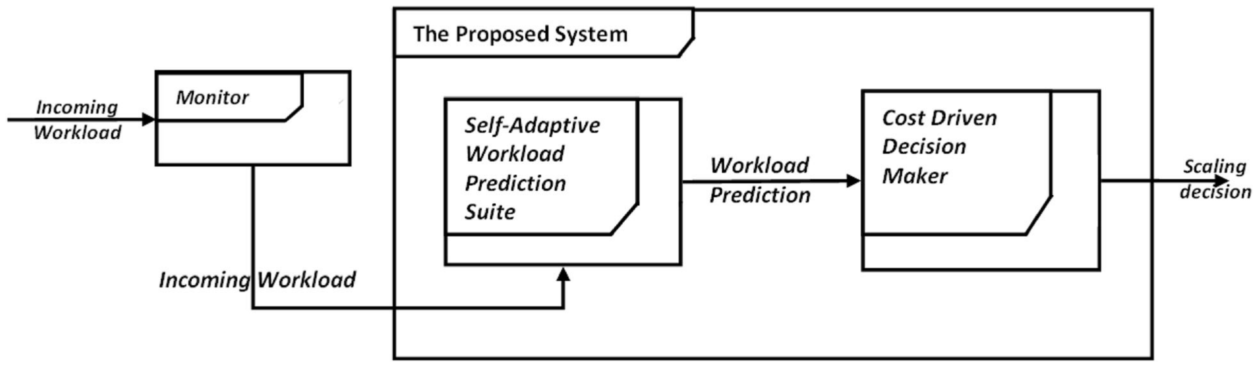

Fig. 2 Predictive Auto-scaling system architecture

- An investigation of the impact of the configuration parameters on the accuracy of the rule-based auto-scaling systems.

The remainder of this paper is organized as follows: section 2.0 discusses the background, the related work, auto-scaling accuracy and cost driven decision maker. In section 3.0, experiments are presented that show the impact of VM boot-up time, configuration parameters, and smart kill on auto-scaling accuracy and cost. This is followed with the proposed optimum configuration for auto-scaling problem using genetic algorithm. The evaluation of the cost driven decision maker and the predictive auto-scaling system is presented in section 5.0. The conclusion and possible future directions for the research are discussed in section 6.0.

Background and related work

In this section we present an overview of the existing auto-scaling systems, and describe the rule-based auto-scaling technique and introduce its configuration parameters. In addition, we summarize our previous work [6] on self-adaptive prediction auto-scaling suite. This summary is necessary for understanding the present research work in this paper. The authors in [3] group the existing auto-scaling approaches into five categories: rule based technique, reinforcement learning, queuing theory, control theory, and time-series analysis. Among these categories, the time-series analysis focuses on the prediction side of the resource provisioning task and is not a “decision making” technique per se. In contrast, the rule-based technique is a pure decision making mechanism while the rest of the auto-scaling categories plays the predicator and the decision maker roles at the same time. The rule based technique is the only approach which is widely used in the commercial auto-scaling systems [7–9].

Existing auto-scaling systems

Auto-scaling systems can be grouped into *reactive* and *predictive* categories. Reactive systems scale-in or -out a cloud service based on the current performance of the cloud service. Reactive systems use either rule-based or schedule-based techniques to carry out the auto-scaling task. Rule-based systems use a set of scaling rules to scale-in or -out a cloud service when its performance reaches a predefined threshold. Schedule-based mechanism allows cloud clients to add or remove VMs at a given time and are suitable when the changes in the workload are known ahead of time [10]. However, not all of the cloud services have time-based workload patterns, and it is not straightforward for the cloud clients to correctly determine all the related scaling indicators or the thresholds based on the performance goals [10].

Predictive auto-scaling systems forecast the cloud service’s future workload and adjust the compute and the storage capacity in advance to meet the future needs. Predictive auto-scaling systems can be grouped into four categories [3]: reinforcement learning, queuing theory, control theory, and time-series analysis. Among these categories, the time-series analysis focuses on the prediction side of the resource provisioning task and is not a “decision making” technique per se. Therefore, a time-series analysis technique should be bundled with a decision maker to create a predictive auto-scaling system. Queuing theory models each VM as a queue of requests and calculates the performance metrics’ values. The calculated values are used to generate a scale action. Reinforcement learning algorithms handle the auto-scaling task without any à priori knowledge or system model. However, the time for the reinforcement learning methods to converge to an optimal policy can be unfeasibly long. Control theory creates a reactive or a predictive controller to automatically adjusting the required resources to the cloud service’s demand. Readers are encouraged to see [3] for more details about the different decision making approaches.

The proposed auto-scaling system (see Fig. 2) avails the predictive approach to carry out the auto-scaling task. Since time-series analysis is the most dominant prediction technique in the cloud auto-scaling domain [3], the prediction suite uses the time-series analysis technique to forecast the future workload of the cloud service. Moreover, our prediction suite applies *decision fusion* technique [11] to increase the prediction accuracy (see [6] for more details on the prediction suite). The cost driven decision maker uses the rule-based technique to generate the scaling decisions. Although the rule-based technique is easy to use, it is not a trivial task to configure the rule-based systems. The proposed cost-driven decision maker uses the genetic algorithm principle to overcome this problem.

Self-adaptive prediction suite

This subsection summarizes our previous work in [6] which serves as a foundation for the research work in this paper. Researchers have already used prediction methods to alleviate the reactive nature of the rule-based systems. However, the existing predictive auto-scaling systems use only one prediction method to forecast the future performance condition of the cloud service. Therefore, to increase the prediction accuracy, our predictive auto-scaling system identifies the pattern of the incoming workload and chooses the prediction algorithm based on the detected pattern. Therefore, the self-adaptive suite automatically chooses:

- The Multi-layer Perception (MLP) prediction model to forecast the workload in the environments with the unpredictable workload pattern
- The Multi-Layer Perception with Weight Decay (MLPWD) prediction model to forecast the workload in the environments with the periodic workload pattern
- The Support Vector Machine (SVM) prediction model to forecast the workload in the environments with the growing workload pattern

Readers are encouraged to read the paper in [6] for more details about the reasons to choose the aforementioned prediction models and their corresponding environments.

The objective of a classical self-adaptive system is to make the system self-managed as a result of objects change or environmental influence on the inputs. A requirement in this context is that the system must be able to keep knowledge about its past, present, and future goals. In our case, self-adaptive prediction suite architecture is designed by adapting the classical autonomic system architecture to the cloud auto-scaling system (see Fig. 3). The cloud auto-scaling architecture consists of a cloud workload context element; a cloud auto-scaling system which includes the meta-autonomic elements (the workload pattern and the cloud auto-scaling); and a cloud computing scaling decisions element. In addition, an element for the autonomic manager, knowledge, and goals is added to the architecture. The *cloud workload usage* represents the *real world usage context* while the *cloud computing scaling decisions* represents the *computing environment* context. It is important to note that

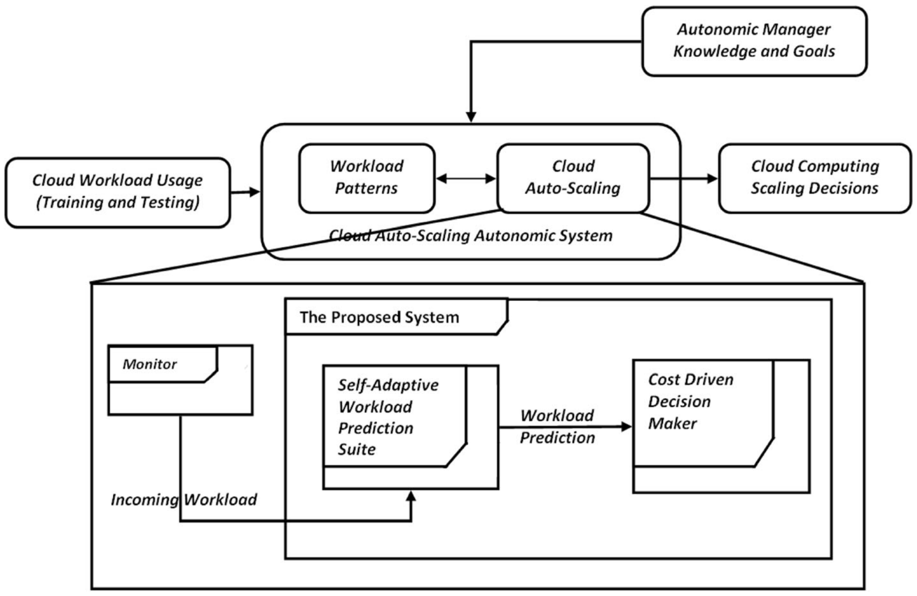
Autonomic Manager Knowledge and Goals
Cloud Workload Usage (Training and Testing)
Workload Patterns
Cloud Auto-Scaling
Cloud Computing Scaling Decisions
Cloud Auto-Scaling Autonomic System
The Proposed System
Monitor
Self-Adaptive Workload Prediction Suite
Workload Prediction
Cost Driven Decision Maker
Incoming Workload

Fig. 3 Projection of the cloud auto-scaling autonomic element

an autonomic system always operates and executes within a context. The context in general is defined by the environment as well as the runtime behavior of the system. The purpose of the autonomic manager is to apply the domain specific knowledge which is linked to the cloud workload pattern and apply the appropriate predictor algorithm (see Fig. 3) to predict the future workload. The cloud autonomic manager is constructed around the analyze/decide/act control loop.

The prediction suite identifies the pattern of the incoming workload and chooses the most accurate prediction algorithm based on the workload pattern. The cloud auto scaling autonomic elements (i.e., the workload patterns and the predictor component) are designed such that the architecture can be implemented using the strategy software design pattern (see Fig. 4). The strategy software design pattern consists of a strategy and a context. In the cloud auto-scaling domain, the predictor is the strategy and the workload pattern is the context. In general the strategy and the context interact to implement the chosen algorithm. A context passes all of the data (i.e., the workload pattern) that is required by the algorithm to the strategy.

Rule-based systems

In the rule-based auto-scaling, the number of the leased VMs varies according to a set of scaling rules. A scaling rule has two parts: the condition and the action to be executed when the condition is met. The condition part of a scaling rule uses one or more performance indicator(s), such as the average response time or the average workload. A typical rule-based system has six configuration parameters: the upper threshold (*thrU*), the lower threshold (*thrL*), the upper scaling duration (*durU*), the lower scaling duration (*durL*), the upper cool-down duration (*inU*), and the lower cool-down duration (*inL*). A performance indicator has an upper (i.e., *thrU*) and a lower (i.e., *thrL*) thresholds. If the scaling condition is met for a given duration (i.e., *durU* or *durL*) then the corresponding action will be triggered. After executing a scale action, the decision maker stops itself for a cool-down period which is defined by *inU* or *inL*.

Some research works have proposed additional parameters to improve the auto-scaling accuracy. For instance, the proposed method in [12] uses two upper and two lower thresholds to determine the trend of the performance indicator. Considering the trend of the performance indicator helps to predict the future performance of the cloud service and generate the scale actions ahead of time. Although the proposed method in [12] generates the scale actions ahead of time, it does not have a better accuracy compared to the traditional rule-based systems [3]. This paper uses a typical rule-based system to scale-in (or -out) the cloud service.

Specification of the auto-scaling accuracy

The auto-scaling accuracy is closely related to the cost incurred by the cloud clients. The more accurate the auto-scaling system, the lower the cost incurred by the cloud clients. Therefore, cost is the main metric that measures the accuracy of the auto-scaling systems. From the cloud client's perspective, there are two types of costs associated with the auto-scaling systems: *resource cost* (C_R) and *SLA violation cost* (C_{SLA}).

Resource cost refers to the cost of the leased VMs and can be measured by the number of the leased VMs and their hourly rental rate. This paper assumes that the IaaS provider supplies only one type of VM with a fixed hourly rate. Then, the resource cost can be measured by:

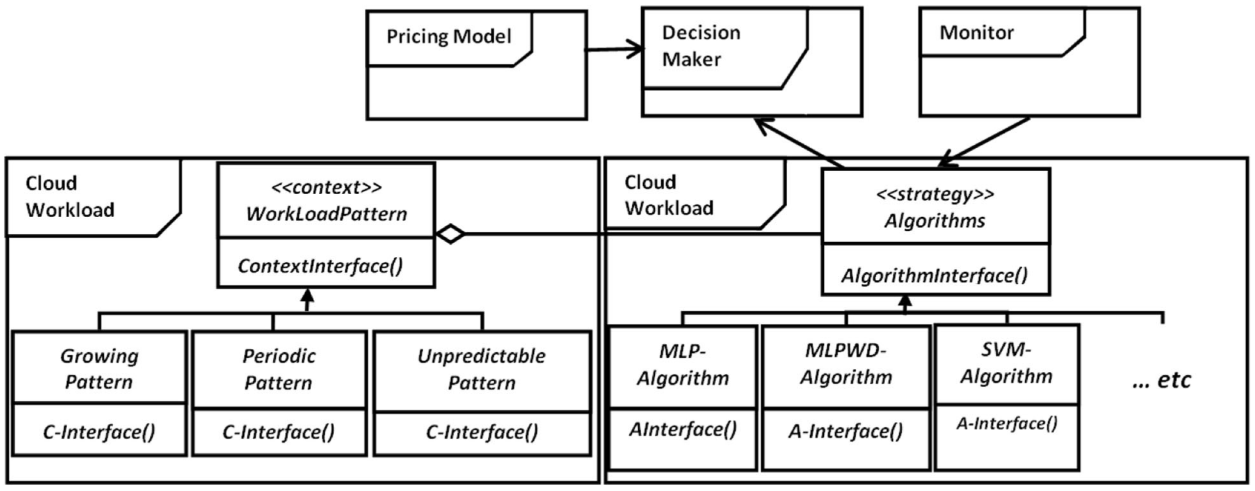

Fig. 4 The design of the autonomic elements using strategy design pattern

$$C_R = \sum\nolimits_{t=0}^{T} n_t \times c_{vm} \tag{1}$$

where T is the total hours that the auto-scaling system is running, c_{vm} is the hourly rate of leasing a VM, and n_t is the number of the leased VMs between hour t and $t + 1$.

SLA violation cost is the cost associated with the SLA breaches. A SLA breach (i.e., SLA violation) refers to any act or behavior that does not comply with the SLAs document. In this paper, *response time* is considered to be the main Quality-of-Service factor and any request with a response time more than the maximum response time (which is defined in the SLAs document) is recognized as a SLA violation. Therefore, total number of SLA violations v_t at time t is defined as:

$$\begin{aligned} v_t &= \sum\nolimits_{req=1}^{N} v_{t,req} \\ v_{t,req} &= \begin{cases} 1 & if\ \left(r_{req} - R\right) > 0 \\ 0 & otherwise \end{cases} \end{aligned} \tag{2}$$

where *req* represents an incoming request, N is the total number of requests at time t, r_{req} is the response time of the request *req*, and R is the maximum response time defined in the SLAs document.

Measuring SLA violation cost depends on different factors, such as the downtime duration of the cloud service, the number of affected end users, and even the sociological aspects of the end users' behaviors. In this paper a constant penalty c_b is assigned to each SLA violation. The value of c_b is defined by the cloud client who provides the cloud service. The SLA violation cost is:

$$C_{SLA} = \sum\nolimits_{t=o}^{T} v_t \times c_b \tag{3}$$

A highly accurate auto-scaling system prevents SLA violations as well as reduces the resource cost. However, it is not possible to minimize the number of the SLA violations and the resource cost at the same time. Adding more infrastructural resources reduces the number of the SLA violations, but results in an excessive resource cost. On the other hand, releasing the infrastructural resources saves the resource cost, but increases the number of the SLA violations. Therefore, the auto-scaling system's job is to find the balance between the SLA violations and the resource cost (i.e., the cost/performance trade-off). The optimum solution to this trade-off varies for the different cloud clients. The smaller businesses that do not have many end users, such as startup companies, usually prefer to reduce the resource cost, while the bigger businesses that have many end users, such as eBay or Netflix, prefer to minimize the SLA violations. Therefore, the cloud client's cost preference is one of the factors that should be considered by the auto-scaling system to solve the cost/performance trade-off.

Specification of the cost-driven decision maker

Recall that a performance indicator has an upper (i.e., *thrU*) and a lower (i.e., *thrL*) threshold. If the scaling condition is met for a given duration (i.e., *durU* or *durL*) then the corresponding action will be triggered. After executing a scale action, the decision maker stops itself for a small cool-down period which is defined by *inU* or *inL*. In order to have an accurate rule-based system it is crucial to configure the system such that the resource cost and SLA violation cost are minimized. However, the resource cost and the SLA violation cost cannot be minimized at the same time and the balance point between them depends on the cloud client's cost preference (see Section 2.4.).

The objective here is to find the best value for each of the configuration parameters such that the configured rule-based decision maker minimizes the final auto-scaling cost. Since the domain of valid values for each of the parameters is known, the universal set of possible solutions can be created where each solution is a valid combination of the parameters. Then to find the optimal solution, the search space (i.e., the universal set of the solutions) is traversed and the solution with the least auto-scaling cost is found. To measure the auto-scaling cost of a given solution, a decision maker is configured with the parameters of that solution, and an auto-scaling simulation is run for a predefined duration to calculate the total auto-scaling cost of the decision maker. In this paper an in-house simulation package [13] is implemented and used to carry out the simulations. Based on the simulation result, measuring the auto-scaling cost of a given solution averagely takes five minutes. Therefore, for a search space with 100 possible solutions, it takes 500 min (more than eight hours) to traverse the search space and find the optimum solution.

For example, assume that in an auto-scaling environment the CPU utilization is considered as the performance indicator. Since the CPU utilization value is always between 0 and 100, the upper threshold can take 100 different values. In addition, the lower threshold can take any value greater than zero and less than the upper threshold. Moreover, suppose that the *inU*, *inL*, *durU*, and *durL* take any values between 0 and 5 min. In this environment, the universal set includes 6,413,904 valid solutions. Given that measuring the total cost of a solution takes 5 min, traversing the whole search space takes 32,069,520 min (i.e., more than 61 years), which is infeasible to perform. Therefore, this paper proposes a genetic algorithm model to find an optimal solution within the search space in a shorter time.

Impact of VM-boot-up time, SMARTKILL and configuration parameters on AUTOSCALING accuracy and costs

In this section we present the result of three experiments on the impact of VM boot-up time on auto-scaling accuracy, and the impact of the smart kill technique on auto-scaling cost factors and the impact of configuration parameters on auto-scaling accuracy. The results of impact of configuration parameters show how difficult it is for a cloud client to handle the configuration parameters. In addition, the smart kill results show how important the smart kill is in decreasing resource cost or reducing SLA violations.

VM boot-up time Vis-à-Vis the auto-scaling accuracy

An in-house simulation package [13] is developed and used to carry out simulation on the effect of VM boot-up time on the cost factors of a rule-based decision maker.

In the simulation, three instances of the cloud workload patterns are sent to a multi-layer cloud service which is deployed to an IaaS infrastructure. In each of the monitoring intervals, the decision maker compares the incoming workload with the capacity of the cloud service. The capacity of the cloud service refers to the number of the requests that cloud service can accommodate per second. If the incoming workload exceeds the cloud service's capacity (i.e., the upper threshold) then the auto-scaling system scales up the infrastructure. If the cloud service's upper threshold is not reached, the auto-scaling system verifies whether the cloud service is still able to accommodate the incoming workload after releasing one of the provisioned VMs (i.e., the lower threshold). If so, then the auto-scaling system scales down the cloud service. Table 1 shows the configuration parameters that are used in the simulation.

According to Table 1 the cloud service has two tiers. The VM boot-up time for both of the tiers is the same (i.e., 10 min). The VM boot-up time in the commercial cloud computing environment is reported to be between 5 and 15 min [14–17]. In addition, the service demand of the business tier and the database tier are identical. Since the goal of this experiment is to measure the impact of the VM boot-up time on the cost factors, the service demand and the database access rate do not affect the result. The service demand and the database access rate values are decided based on the values that are used in similar experiments in [18].

Table 1 Simulation parameters and values

Parameter Name	Value
Number of the cloud service tiers	2
Business tier's service demand	0.076 s
Business tier's access rate	1
Business tier's VM boot-up time	10 min
Database tier's service demand	0.076 s
Database tier's access rate	0.7
Database tier's VM boot-up time	10 min
The monitoring interval	5 min

The experiment compares the number of the SLA violations and the resource cost of three IaaS environment: "*IaaS A*" with a VM boot-up time = 10 min, "*IaaS B*" with a VM boot-up time = 5 min, and "*IaaS C*" with a VM boot-up time = 0 min. The experiment has two iterations. In the first iteration the cloud service is simplified and it is assumed that the cloud service consists of just a business tier and there is no database tier. In the second iteration, a more common scenario is investigated in which, the cloud service consists of a business tier as well as a database tier. Tables 2 and 3 present a comparison of the SLA violation count and the resource cost of the *IaaS A, IaaS B,* and the *IaaS C* for the first and the second iterations, respectively. In the experiment, it is assumed that $c_{vm} = 1\ \$/hr$ (i.e., the leasing rate of a VM is $1 per hour).

According to the results, by reducing the VM boot-up time from 10 min to five minutes, and to zero minute, the cost factor (i.e., the resource cost and the SLA violation count) decreases. Comparing the cost factors of the *IaaS A* with the *IaaS C* shows that decreasing the VM boot-up time from 10 min to 0 min causes:

- A significant decrease in the SLA violations
- Iteration 1: the SLA violations for the periodic, growing, and unpredictable workloads are reduced by 92.7%, 97.9%, and 87.8%, respectively.
- Iteration 2: the SLA violations for the periodic, growing, and unpredictable workloads are reduced by 86.8%, 87.5%, and 86.9%, respectively.
- A decrease in the resource cost
- Iteration 1: the resource cost of the periodic, growing, and unpredictable workloads are reduced by 19.3%, 19%, and 17%, respectively.
- Iteration 2: the resource cost of the periodic, growing, and unpredictable workloads are reduced by 18.6%, 15.6%, and 17.3%, respectively.

Since the VM boot-up time is a characteristic of the IaaS environment, in the real world, the VM boot-up time cannot be reduced to zero. Therefore, researchers have used the predictive auto-scaling approach to imitate reducing the VM boot-up time. For instance, assume the VM

Table 2 The cost factor values for the different VM boot-up times (iteration 1 – no database tier)

Workload Type	Resource cost			SLA violation count		
	IaaS A	IaaS B	IaaS C	IaaS A	IaaS B	IaaS C
Periodic	$192	$172	$155	83	14	6
Growing	$147	$138	$119	47	8	1
Unpredictable	$182	$173	$151	90	23	11

Table 3 The cost factor values for the different VM boot-up times (iteration 2 – with database tier**)**

Workload type	Resource cost			SLA violation count		
	IaaS A	IaaS B	IaaS C	IaaS A	IaaS B	IaaS C
Periodic	$333	$301	$271	106	28	14
Growing	$243	$219	$205	72	18	9
Unpredictable	$318	$288	$263	107	39	14

boot-up time in a given IaaS environment is 10 min. In that environment, a prediction algorithm can be used to forecast the workload of the cloud service 10 min ahead of time. This allows the decision maker to generate the scaling decisions based on the prediction results. Therefore, to scale up the infrastructure, a scaling action gets generated 10 min ahead of time, and as a result, the new VM will be ready in time. This is practically equivalent to reducing the VM boot-up time to zero minute. It should be noted that the prediction cost in the auto-scaling systems is negligible.

The smart kill effect on the auto-scaling cost factors

The smart kill technique suggests not killing the VMs before they are used for a full hour. The reason is that the IaaS providers round up the partially used hours to the full hours and charge the cloud clients for a full hour VM usage, even if the VM is being used for less than an hour. Therefore, we conducted an experiment to compare the number of SLA violations and resource cost of two rule-based systems: *System A* which uses the smart kill, and *System B* which does not use the smart kill. The configuration parameters of the rule-based decision maker are presented in Table 4. Furthermore, in this simulation the target cloud service has two tiers and the VM boot-up time is assumed to be zero minutes. Table 5 shows the results while Table 4 shows the parameters that was used in the experiment.

According to the results, using the smart kill:

- Slightly decrease the resource cost: periodic workload 2.2%, growing workload 0.5%, and unpredictable workload 1.5%

Table 4 The parameters of the rule-based decision maker

Parameter Name	Value
thrU	ceiling capacity
thrL	floor capacity
durU	0 min
durL	0 min
inU	0 min
inL	0 min

Table 5 The impact of the smart kill on the cost factors

Workload type	Operational cost		SLA violation count	
	Smart kill	No smart kill	Smart kill	No smart kill
Periodic	$265	$271	5	14
Growing	$204	$205	2	9
Unpredictable	$259	$263	5	14

- Reduces the SLA violations: periodic workload 64.3%, growing workload 77.8%, and unpredictable workload 64.3%

The reason why the smart kill decreases the SLA violations is that the smart kill postpones the scale in actions to the end of the VM's billing hour [19, 20]. This increases the capacity of the cloud service and prevents some of the SLA violations that occur due to the unexpected workload surge [21]. In addition, the smart kill reduces the resource cost, because according to the Amazon AWS pricing model [22], if the cloud clients stop and start the exact same instance 5 times in an hour, they get billed for 5 h. The approaches that do not use the smart kill stop and start the VMs more frequently use averagely more VMs during each hour period, which increases the resource cost. As a result of this outcome, smart kill is used in our decision maker algorithm.

Impact of the configuration parameters on the auto-scaling accuracy

This section investigates the impact of the configuration parameters of a rule-based system on the cost factors. The purpose of these experiments is to demonstrate the need for the use of genetic algorithm (or any other algorithm) not just to reduce the space search time but also to find optimum configuration for the different parameters. Note that our rule-based decision makers have six configuration parameters: *thrU, thrL, durU, durL, inU*, and *inL*.

The upper threshold (thrU)

The first iteration analyses the influence of the *thrU* on the SLA violations and the resource cost. The values of the other configuration parameters (except the *thrU*) are held constant during the simulation to isolate the relationship between the *thrU* and the cost factors. In this experiment a multi-tier cloud service is simulated. According to Table 1, the business and the database tiers have the same service demand which is 0.076 s. This value is decided according to the service demand that is used in [14]. Based on the QNM (Queuing Network Model) theory [19, 23–26], each of these tiers can handle up to $\frac{1}{0.076} \approx 13$ requests per second. Therefore, to fully utilize each of the VMs capacities, the *thrU* should be set to 13 requests per second. This way, the cloud service can accommodate up to $13 \times n$ requests per second, where n is the number of the leased VMs. To test the relationship between the *thrU* and the cost factors, the *thrU* value is decreased slowly and the number of the SLA violations and the resource cost are measured.

According to the results (see Table 6), decreasing the upper threshold increases the resource cost, while it reduces the number of the SLA violations. Since the upper threshold indicates the capacity of the leased VMs, decreasing the upper threshold prevents the VMs from becoming fully utilized by increasing the spare capacity of each of the VMs. Increasing the spare capacity of the VMs necessitates renting more resources thus increasing the resource cost. However, because the leased VMs are not fully utilized, they are able to accommodate the unforeseen workload surges; therefore, the number of the SLA violations reduces.

The lower threshold (thrL)

The second iteration of the experiment investigates the impact of the *thrL* on the SLA violations and the resource cost. The floor capacity (see eq. 9 in section 4.5) represents the lower threshold. The decision maker scales down the cloud service whenever the incoming workload becomes less than the floor capacity. The floor capacity calculates the maximum capacity of the cloud service after releasing one of the leased VMs, and indicates the earliest time the decision maker can release one of the VMs without increasing the risk of under-provisioning. Table 7 indicates that, although increasing the lower threshold increases the SLA violations, it does not significantly reduce the resource cost. Table 8 represents the result of decreasing the thrL. Based on eq. 9 (section 4.5), decreasing the lower threshold postpones the scale in action and thus increases the resource cost. On the other hand, decreasing the thrL (see Table 8) reduces the SLA violations (except for the unpredictable workload) because the decision maker releases a VM only if the remaining VMs have more than enough capacity to handle the incoming workload. However, in the environments with the unpredictable workload the number of the incoming requests may change dramatically between the monitoring intervals.

Theoretically, increasing the floor capacity accelerates the scaled-in action and reduces the resource cost. On the other hand, increasing the lower threshold (i.e., the floor capacity) can increase the risk of the SLA violations, because it makes the decision maker to release one VM before the remaining VMs' capacities are enough to handle the incoming workload. The results (Table 7) indicate that, although increasing the lower threshold (thrL) increases the SLA violations, but, this does not significantly reduce the resource cost (C_R). Also, decreasing thrL (Table 8) decreases SLA violations but does not significantly impact cost (C_R).

The duration parameters (durU and durL)

In the third iteration of the experiment, the influence of the durL and the durU parameters on the resource cost and the SLA violations are measured. The results for durU and durL are shown in Tables 9 and 10 respectively. Similar to reducing the *thrL*, increasing the *durU* postpones the scaled-out action which reduces the resource cost and has a significant negative influence on the SLA violations. In addition, increasing the *durL* postpones the scaled-in action, thus increasing the resource cost and reducing the SLA violations. However, the number of the SLA violations does not significantly decrease by increasing the *durL*. Therefore, one can suggest the use of a smaller *durL* value to reduce the overall auto-scaling cost.

The freezing durations (inU and inL)

The last iteration of the experiment studies the impacts of the *inU* and the *inL* parameters on the cost factors (cf. Tables 11 and 12). In theory, increasing the *inU* postpones the scaled-out actions which should increase the SLA violations and reduce the resource cost.

Table 6 Impact of decreasing thrU on the cost factors

thrU	Periodic		Growing		Unpredictable	
	C_R ($)	SLA violations	C_R ($)	SLA violations	C_R ($)	SLA violations
13	265	5	204	2	259	5
12.5	270	4	210	2	263	5
12	284	4	214	1	271	3
11.5	287	5	221	1	277	2
11	295	1	229	0	282	2
10.5	304	0	234	0	294	1
10	311	0	242	0	310	0

Table 7 Impact of increasing thrL on the cost factors

thrL	Periodic		Growing		Unpredictable	
	C_R ($)	SLA violations	C_R ($)	SLA violations	C_R ($)	SLA violations
13	265	5	204	2	259	5
13.5	264	24	202	28	257	17
14	260	50	201	31	253	35
14.5	260	61	201	35	252	49
15	259	71	199	41	250	63
15.5	255	89	199	57	248	70
16	253	95	198	61	248	77

Similarly, increasing the *inL* postpones the scaled-in actions which should reduce the SLA violations and increase the resource cost. However, according to the experimental results, the parameters *inU* and *inL* do not significantly impact the cost factors. This is because the auto-scaling model uses the smart kill technique which postpones the scaled-in action. Therefore, since the scaled-in actions are already frozen by the smart kill technique, increasing the *durL* does not change the cost factors much.

Summary

The results in this section suggest that all of the configuration parameters can influence the SLA violations and the resource cost of the rule-based decision maker. However, some of the parameters, such as the *thrU* and the *durU*, have more influence on the cost factors.

The experimental results show that to decrease the resource cost, cloud clients should:

- Increase the upper threshold (*thrU*)
- Increase the lower threshold (*thrL*)
- Increase the upper scaling duration (*durU*)
- Decrease the lower scaling duration (*durL*)
- Increase the upper freezing duration (*inU*)
- Decrease the lower freezing duration (*inL*)

On the other hand, to reduce the SLA violations, cloud clients should:

- Decrease the upper threshold (*thrU*)
- Decrease the lower threshold (*thrL*)
- Decrease the upper scaling duration (*durU*)
- Increase the lower scaling duration (*durL*)
- Decrease the upper freezing duration (*inU*)
- Increase the lower freezing duration (inL)

In conclusion, the question here is "By how much should the cloud client decreases or increases a configuration parameter?" This question can only be answered if we can find optimum configuration for the different configuration parameters – hence the use of genetic algorithm.

Finding an optimum configuration for AUTOSCALING problem

The genetic algorithm (GA) applies the evolution principle to provide a robust search technique that finds a high-quality solution in a large search space in polynomial time. A genetic algorithm combines the exploitation of the best solutions from the past searches with the exploration of the new regions of the solution space [27]. Any solution in the search space is represented by a chromosome.

Table 8 Impact of decreasing thrL on the cost factors

thrL	Periodic		Growing		Unpredictable	
	C_R ($)	SLA violations	C_R ($)	SLA violations	C_R ($)	SLA violations
13	265	5	204	2	259	5
12.5	266	4	207	1	262	4
12	266	4	207	1	261	3
11.5	267	4	209	1	266	4
11	269	3	213	1	271	3
10.5	270	3	215	1	276	2
10	270	3	216	1	280	3

Table 9 Impact of increasing durU on the cost factors

durU	Periodic		Growing		Unpredictable	
	C_R ($)	SLA violations	C_R ($)	SLA violations	C_R ($)	SLA violations
0	265	5	204	2	259	5
1	255	62	199	48	241	71
2	248	106	196	86	233	112
3	247	134	187	123	229	137
4	236	177	183	148	220	173
5	234	197	180	161	220	174

Essentially, in this research work, there is a need to balance the "resource cost" and the "SLA violation" vis-à-vis the six rule-based configuration parameters: the upper threshold (*thrU*), the lower threshold (*thrL*), the upper scaling duration (*durU*), the lower scaling duration (*durL*), the upper cool-down duration (*inU*), and the lower cool-down duration (*inL*). These parameters lend themselves to evolutionary behavior (see sections 2.3, 2.4, and 2.5) which can be adequately captured by genetic algorithms (GAs).

A genetic algorithm maintains a population of the chromosomes that evolves over the generations (i.e. iterations). The quality of a chromosome in the population is determined by a fitness function. The fitness value indicates how good a chromosome is compared to the other chromosomes in the population [27]. A typical genetic algorithm has the following steps:

1. Create an initial population consisting of randomly generated solutions.
2. Generate a new offspring by applying the genetic operators which are: selection, crossover and mutation, one after the other.
3. Calculate the fitness value of the chromosomes.
4. Repeat steps 2 and 3 until the algorithm converges to an optimal solution.

In order to use the genetic algorithm principle to solve the auto-scaling problem, the representation of the chromosomes in the population, the fitness function, and the genetic operators should be determined.

Chromosome representation

In the rule-based systems, a feasible solution is required to meet the following conditions:

- The upper threshold (thrU) should be less than or equal to the upper limit of the performance indicator. For instance, if CPU utilization is the performance indicator, the upper threshold cannot be greater than 100%.
- The lower threshold (thrL) should be greater than or equal to the lower limit of the performance indicator. For instance, if CPU utilization is the performance indicator, the lower threshold cannot be less than 0%.
- The freezing periods should be greater than or equal to zero (i.e., $inU \geq 0$, $inL \geq 0$).
- The duration parameters should be greater than or equal to zero (i.e., $durU \geq 0$, $durL \geq 0$).
- The upper threshold should be greater than the lower threshold (i.e., $thrU > thrL$).

A chromosome in the population is a feasible set of the configuration parameters, and has the following format:

$$< thrU, thrL, durU, durL, inU, inL > \tag{4}$$

Table 10 Impact of increasing the durL on the cost factors

durL	Periodic		Growing		Unpredictable	
	C_R ($)	SLA violations	C_R ($)	SLA violations	C_R ($)	SLA violations
0	265	5	204	2	259	5
1	271	5	208	2	263	5
2	280	3	211	2	268	3
3	284	3	214	1	271	2
4	291	3	217	1	276	2
5	294	3	219	1	280	0

Table 11 Impact of increasing inU on the cost factors

inU	Periodic		Growing		Unpredictable	
	C_R (\$)	SLA violations	C_R (\$)	SLA violations	C_R (\$)	SLA violations
0	265	5	204	2	259	5
1	259	5	204	4	259	5
2	259	5	203	11	259	5
3	259	7	201	22	259	7
4	259	7	197	45	259	7
5	257	15	197	47	257	19

Each of the configuration parameters is referred to as a “gene”. In the auto-scaling problem, a gene represents a value of one of the configuration parameters of the rule-based decision maker.

Fitness function
A fitness function is used to measure the quality of the chromosomes in the population based on a given optimization objective. Since the goal of the decision maker is to reduce the total cost, the fitness function measures the total auto-scaling cost of the chromosomes. The total cost of a chromosome C_{total} (eq. 5) is the summation of its SLA violations cost C_{SLA} and its resource cost C_R:

$$C_{total} = C_R + C_{SLA} = \sum_{t=o}^{T} \sum_{req=1}^{N} (c_b \times v_{t,req}) + \sum_{t=0}^{T} n_t \times c_{vm} \tag{5}$$

The values of the constant parameters (i.e., c_b and c_{vm}) are determined by the cloud clients. This way, the cloud clients can change the fitness function base on their cost preferences.

Genetic operators
The genetic operators manipulate the chromosomes in the current population and generate a new population of chromosomes. A genetic algorithm has three operators:

Table 12 Impact of increasing inL on the cost factors

inL	Periodic		Growing		Unpredictable	
	C_R (\$)	SLA violations	C_R (\$)	SLA violations	C_R (\$)	SLA violations
0	265	5	204	2	259	5
1	265	5	206	2	259	5
2	265	4	208	2	259	4
3	265	4	210	2	259	4
4	265	4	213	2	259	4
5	270	4	213	2	260	4

- *Selection operator* equates to the survival of the fittest and gives preference to better chromosomes to pass on their genes to the next generation. In the auto-scaling problem, the selection operation finds the sets of the configuration parameters (i.e., the chromosomes) with the least cost in passing their parameter values (i.e., the genes) to the next generation. In this paper, a combination of the *elitism* [28] and the *roulette wheel* [28] selection methods is used. This combination guarantees to keep the chromosomes with the highest fitness values in the next generation.
- *Crossover operator* represents the mating between chromosomes. In the auto-scaling problem, a crossover operation represents how the selected sets of the configuration parameters are combined to create a new set of the configuration parameters. The type and implementation of the crossover and the mutation operators depend on the encoding of the individuals. In our case, the *value encoding* is used. The crossover types for the value encoding are: Single-point crossover; Two-point crossover; Uniform crossover; and Arithmetic crossover. In the auto-scaling problem, there are some limitations on the values of the genes. For instance, the lower threshold (*thrL*) should always be less than the upper threshold (*thrU*). Since the gene values depend on each other, using the single-point or the two-point crossover methods can produce an invalid offspring. Therefore, the *uniform crossover* method [28] is used in this paper to create a new generation.
- *Mutation operator* introduces the random modifications in the genes of the chromosomes. The purpose of the mutation operator is to maintain the diversity in the population and avoid a premature convergence. For the value encoding, the mutation operator is implemented by adding a small number to the randomly selected genes. In the auto-scaling problem domain one of the genes (i.e., a configuration parameter) is randomly picked and its value is altered.

Finding the optimal values for the genetic algorithm
The accuracy of the rule-based algorithm is defined by the auto-scaling cost which is related to the set of the configuration parameters that is found by the genetic algorithm.

In this experiment, to calculate the fitness value of an individual (each individual represents a valid combination of the rule-based configuration parameters), the genetic algorithm uses the individual’s genes values to configure a rule-based decision maker and measures the total auto-scaling cost that the rule-based decision maker incurs to the cloud clients. The genetic algorithm

Table 13 Impact of population size on the genetic algorithm accuracy

Configuration				Result	
Population size	No. of generations	Crossover rate	Mutation rate	Fitness	Duration
10	6	0.95	0.015	176	657,037
15	6	0.95	0.015	170	858,748
20	6	0.95	0.015	167	1,182,530
25	6	0.95	0.015	166	1,488,723
30	6	0.95	0.015	164	1,881,464
35	6	0.95	0.015	158	2,103,345
40	6	0.95	0.015	158	2,324,168
45	6	0.95	0.015	157	2,787,369
50	6	0.95	0.015	158	3,022,506
55	6	0.95	0.015	156	3,297,244
60	6	0.95	0.015	158	3,678,164
65	6	0.95	0.015	158	4,159,851
70	6	0.95	0.015	157	4,378,326
75	6	0.95	0.015	157	4,724,120
80	6	0.95	0.015	157	5,057,848

has four configuration parameters: *population size, stop condition, crossover rate,* and *mutation rate.*

Population size

The population size indicates the number of solutions that are examined in each of the iterations of the genetic algorithm. The population should be big enough to cover the diversity of the search space. On the other hand, the larger population sizes increase the duration of the experiment. In the first iteration of the experiment, the relationship between the population size and the accuracy of the genetic algorithm is investigated. The fitness value and the duration of the experiment for the different population sizes are shown in Table 13. Since the initial population in the first iteration of the genetic algorithm is randomly generated (see step 3, Algorithm 1 in section 4.5), the experiment is repeated five times and the results are averaged to increase the precision of the experiment. Figure 5 shows the fitness value to the population size, and the duration to the population size relationships.

According to the results, increasing the population size increases the experiment duration and improves the fitness value. However, increasing the population size to more than 35 individuals does not have a significant impact on the fitness value. Therefore, the optimal population size for the genetic algorithm (i.e. Algorithm 1) in the cloud auto-scaling domain is 35 individuals.

Stop condition

There are three termination conditions that are used in the genetic algorithms: (1) when an upper limit on the number of the generations is reached, or (2) when an upper limit on the number of the evaluations of the fitness function is reached, or (3) when the chance of achieving a significant change in the next generations is excessively low [29]. In this paper the upper limit on the number of generations is used as the stop condition. The reason is that this approach is easy to use and is popular in similar domains [29]. To find the optimal number of the generations, an experiment is carried out to investigate the relationship between the number of generations and the fitness value of the genetic algorithm. Table 14 and Fig. 6 show the results.

Increasing the number of the generations increases the experiment duration and improves the fitness value (i.e., decreases the total cost). However, the fitness value does not significantly improve after 7 generations (see Fig. 6.). Therefore, the optimal number of the generations for the genetic algorithm in the cloud auto-scaling domain is 7 generations.

Crossover rate

The crossover rate or the crossover probability indicates a ratio of how many couples will be selected to generate a new offspring. An ideal genetic algorithm accomplishes a proper balance between the exploration and the exploitation of the search space [28]. The

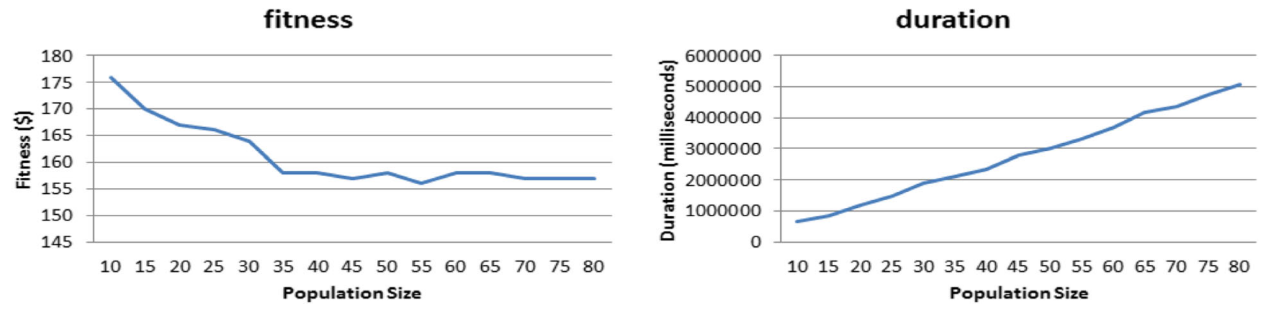

Fig. 5 The population size effect on the experiment duration and the fitness value

Table 14 The impact of the number of the generations on the genetic algorithm accuracy

Configuration				Result	
Population size	No. of generations	Crossover rate	Mutation rate	Fitness	Duration
35	3	0.95	0.015	172	1,161,147
35	4	0.95	0.015	165	1,315,245
35	5	0.95	0.015	160	1,934,609
35	6	0.95	0.015	158	2,109,607
35	7	0.95	0.015	156	2,461,951
35	8	0.95	0.015	156	3,197,381
35	9	0.95	0.015	156	3,305,716
35	10	0.95	0.015	155	3,501,466
35	11	0.95	0.015	157	3,889,925
35	12	0.95	0.015	156	4,180,724
35	13	0.95	0.015	155	4,606,840
35	14	0.95	0.015	155	5,354,189
35	15	0.95	0.015	156	5,780,868
35	16	0.95	0.015	155	5,996,488
35	17	0.95	0.015	155	6,143,828

exploration means searching the search space as much as possible, while the exploitation means concentrating on the global optimum point. In the genetic algorithm, the crossover operator is used to lead the population to converge to the good solutions (i.e., the exploitation). Furthermore, the mutation operators are mostly used to provide the exploration.

The higher crossover rates help to concentrate more on the good solutions in the population. However, if the good solutions are not close to the global optimum solution (i.e., the good solutions are close to a local optimum), then the genetic algorithm cannot find the global optimum solution. This experiment explores the relationship between the crossover rate and the fitness value of the genetic algorithm.

According to the results, increasing the crossover rate increases the experiment duration (see Table 15 and Fig. 7). This is because increasing the crossover rate increases the number of new offspring in each generation, and it takes more time to calculate the fitness of the new offspring. On the other hand, increasing the crossover rate improves the fitness value (i.e., reduces the auto-scaling cost), because the higher crossover rates increase the exploitation which causes the genetic algorithm to converge to the optimum solution. According to the results, the crossover rate of 0.95 is used in the genetic algorithm to create the cost driven decision maker.

Mutation rate

As mentioned in Section 4.3, the mutation operator relates to the exploration function of the genetic algorithm. While the crossover operator tries to converge to a specific point in the search space, the mutation operator avoids the convergence and explores more areas. This helps to prevent from converging around a local optimum point and helps to discover the global optimum solution. However, a high mutation rate increases the probability of searching more areas in the search space and prevents the population to converge to any optimum solution. In other words, a high mutation rate reduces the search ability of the genetic algorithm to a simple random walk while a small mutation rate fails to a local optimum [28]. In this experiment we investigate the impact of the mutation rate on the accuracy of the genetic algorithm. Table 16 shows the impact of the mutation rate on the evolution of the results in different generations. In the experiment, the population size is 35 individuals, and the crossover rate is 0.95.

According to the results, increasing the mutation rate increases the exploration power of the genetic algorithm. However, as shown in Table 16, by increasing the mutation rate, the fitness value does not evolve much in the different generations. This is because the higher mutation rates prevent the algorithm from converging to a optimum point.

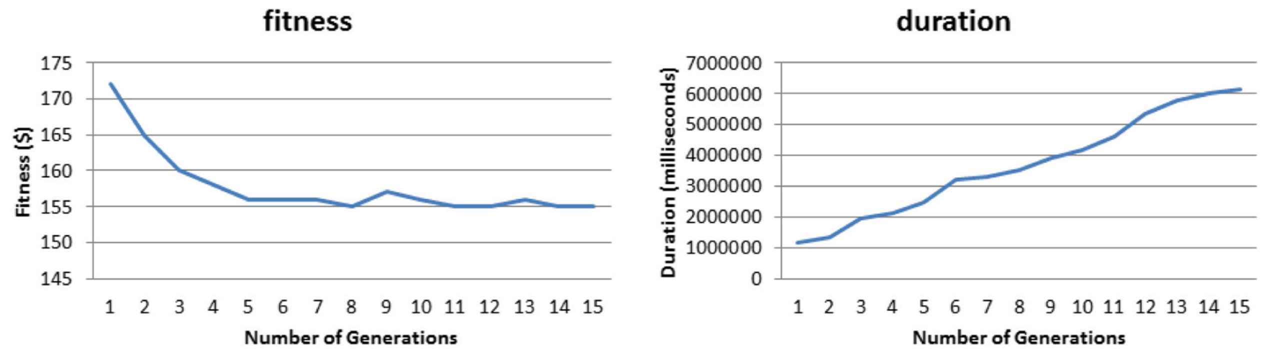

Fig. 6 The number of generation's effect on the fitness value and the experiment duration

Table 15 The impact of the crossover rate on the genetic algorithm accuracy

Configuration				Result	
Population size	No. of generations	Crossover rate	Mutation rate	Fitness	Duration
35	7	0.50	0.015	176	1,763,287
35	7	0.55	0.015	173	1,816,652
35	7	0.60	0.015	170	2,021,755
35	7	0.65	0.015	165	2,164,869
35	7	0.70	0.015	161	2,267,017
35	7	0.75	0.015	160	2,398,733
35	7	0.80	0.015	158	2,567,368
35	7	0.85	0.015	157	2,628,260
35	7	0.90	0.015	155	2,840,778
35	7	0.95	0.015	155	3,070,889

Therefore, mutation rate = 0.02 is used in the genetic algorithm to create the cost driven decision maker.

The proposed decision maker algorithm

This section presents two algorithms. Algorithm 1 uses the genetic algorithm principle to configure the rule-based decision maker. Algorithm 2 shows how the configured decision maker generates the auto-scaling actions.

To fully implement Algorithm 1, we need appropriate values for the parameters: *population size*, a *crossover rate*, a *mutation rate*, and a *stop condition*. These parameters are problem specific, and there is no best global value for these parameters. In section 4.4, we conducted experiments to find values for these parameters and based on the results of the experiments the following values are used to implement the genetic algorithm (i.e. Algorithm 1).

- Population size: 35 chromosomes
- Crossover rate: 0.95 of the population size
- Mutation rate: 0.02 of the population size
- Stop condition: stop the algorithm after seven generations

```
Algorithm 1: The genetic algorithm to configure the decision maker
Inputs: SLA violation cost factor ($C_b$) and VM hourly rate ($C_{vm}$)
Output: The optimal parameter set $< thrU, thrL, durU, durL, inU, inL >$
Begin
   1. Select $C_b$ and $C_{vm}$ values and Formulate the fitness function using $C_b$ and $C_{vm}$
   2. Generate a random population of N individuals based on $< thrU, thrL, durU, durL, inU, inL >$
   3. Calculate the Fitness function $C_{total} = C_R + C_{SLA} = \sum_{t=o}^{T} \sum_{req=1}^{N} (c_b \times v_{t,req}) + \sum_{t=0}^{T} n_t \times c_{vm}$
   4. Select some parents for crossover operation:
            Use uniform crossover method to generate two new offspring.
            Select few individuals randomly for mutation operation. /*The number of genes that are
                  modified during the mutation is defined by the mutation rate. */
            Use elitism selection method to remove individuals with least fitness value from the population
            Copy the remaining individuals to the next generation. /*The number of removed individuals is
                  equal to the number of new offspring, which is defined by the crossover rate.*/
   5. If the experiment duration is over then stop else go to step 3.
   End Begin
```

```
Algorithm 2: The decision maker algorithm
Inputs:  Incoming workload prediction ($\lambda$); Average service demand of each request on the VMs (D); and
         Number of rented VMs
Output: Scaling actions: scale out; scale in; no scaling
Begin
      1. Accept incoming workload prediction from the predictor.
      2. Calculates the ceiling capacity $Cap_{ceiling} = n \times CapU_{vm}$ and floor capacity $Cap_{floor} = n \times$
         $CapL_{vm}$ of the underlying IaaS
      3. If the predicted incoming workload is greater than the ceiling capacity for durU duration
            then generates scale out action for inU duration and go back to step 1

      4. If the predicted incoming workload is less than the floor capacity for durL duration and smart kill
         condition applies
            then generates scale in action for inL duration, and go back to step 1.

      5. If neither scale in nor scale out condition is valid,
            then generates no scale action and go back to step 1.

End Begin
```

The decision maker algorithm (Algorithm 2) uses the *ceiling capacity* and the *floor capacity* concepts. The ceiling capacity ($Cap_{ceiling}$) refers to the maximum number of requests that can be handled by the cloud service, and is calculated as:

$$Cap_{ceiling} = n \times CapU_{vm} \tag{6}$$

Where, n is the number of the provisioned VMs and $CapU_{vm}$ is the upper threshold of the number of the requests that can be handled by each of the VMs per second. According to the QNM (Queuing Network Model) theory, utilization of a VM is calculated as [18, 23, 24, 26]:

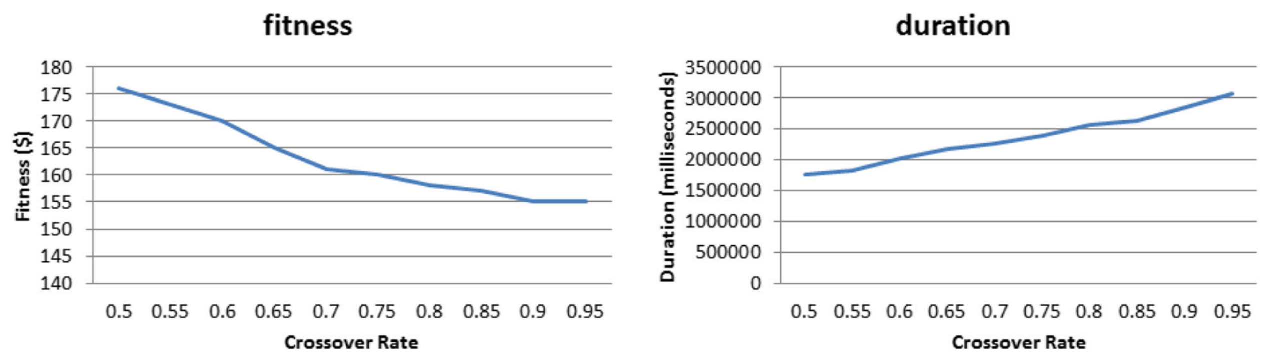

Fig. 7 The crossover rate's effect on the fitness value and the experiment duration

Table 16 The impact of the mutation rate on the genetic algorithm accuracy

Mutation rate	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1	0.11
Generation										
1	175	172	173	169	171	173	171	169	158	158
2	169	165	173	162	171	173	171	169	157	158
3	169	159	156	162	171	171	171	169	157	158
4	158	159	156	162	170	171	171	168	157	158
5	158	159	156	161	168	168	169	168	157	158
6	158	159	154	161	167	168	169	168	157	158
7	158	159	154	161	167	168	168	166	157	158

$$U_{VM}(\lambda) = \lambda \times D_{VM} \tag{7}$$

Where, λ is the incoming workload and D_{VM} is the service demand of the requests on the VM. Given that D_{VM} is static during the experiments (i.e., the service demand of the requests does not change), the maximum number of the requests that can be handled by a VM (i.e. $CapU_{vm}$) is then defined as:

$$\lambda_{max} = CapU_{vm} = \frac{U_{VM_{MAX}}}{D_{VM}} = \frac{thrU}{} D_{VM} \tag{8}$$

The floor capacity (Cap_{floor}) represents the minimum capacity of the cloud service defined as:

$$Cap_{floor} = n \times CapL_{vm} \tag{9}$$

Where, $CapL_{vm}$ is the lower threshold of the number of the requests that can be handled by a VM per second, and is calculated as:

$$\lambda_{min} = CapL_{vm} = \frac{U_{VM_{MIN}}}{D_{VM}} = \frac{thrL}{} D_{VM} \tag{10}$$

The workload should always be bounded by the ceiling and the floor capacities. If the workload exceeds the ceiling capacity (i.e., under-provisioning condition), then a SLA violation occurs. In addition, if the workload is less than the floor capacity (i.e., over-provisioning condition) then, an excessive amount of VMs is being used.

In addition, according to step 4 of the decision maker algorithm (i.e. Algorithm 2), the decision maker scales in the cloud service only if the *smart kill* condition is valid. The smart kill reduces the total number of the SLA violations as well as the resource cost [19].

Complexities of Algorithms 1 and 2

The complexity of GAs has probabilistic convergence time [30, 31]. However, in our case, we have experimentally (see section 4) the population size (i.e. the number of chromosomes) to be 35; the cross over rate is 0.95 of the population; the mutation rate is 0.02 of the population; and the stop condition is set to seven generations. As a result our GA is simple and from experimental results (see section 5) it converges quickly. Therefore, if the number of chromosomes does not have to change every time the instance size increases [30, 31], then the GA complexity is less than exponential time. So, in our case, the instance size is a constant and the selection is done randomly within the same population size and therefore, the best convergence time is linear. From the forgoing, the complexities of Algorithms 1 and 2 in the best case scenario is linear i.e. O(N) and in the worst case scenario is logarithmic i.e. O (log (N)) where N is the population size (i.e. number of chromosomes). Please note that GAs complexity analysis is beyond the scope of this work and readers interested in the theoretical analysis can check reference [31] for detailed study of the subject matter.

Evaluation of the COST-driven decision maker and the auto-scaling system

Evaluation of the cost-driven decision maker

The proposed genetic algorithm is experimentally evaluated to see if the algorithm can identify an optimal configuration using different workload patterns and different client's cost preferences. Tables 17, 18, and 19 show the cost driven genetic algorithm's results in the environments with the periodic, growing, and unpredictable workload patterns respectively. In the tables C_R and C_{SLA} represent the resource and the SLA violation costs, respectively and the optional configuration mapped to <*thrU, thrL, durU, durL, inU,* and *inL* > as defined in eq. 4.

Table 17 The genetic algorithm behavior in an environment with the periodic workload

Iteration no.	C_R	C_{SLA}	No. of rented VMs	No. of SLA breaches	Optimal Configuration
1	1	1	157	4	< 92, 68, 0, 3, 1, 0>
2	1	5	157	4	< 92, 68, 0, 3, 1, 0>
3	1	10	171	2	< 90, 65, 0, 4, 1, 0>
4	1	15	189	1	< 82, 59, 0, 4, 0, 1>
5	1	20	192	0	< 80, 55, 0, 4, 0, 1>
6	1	25	192	0	< 80, 55, 0, 4, 0, 4>
7	1	30	192	0	< 80, 55, 0, 4, 0, 4>
8	5	1	131	59	< 96, 49, 4, 0, 3, 0>
9	10	1	126	84	< 97, 56, 4, 0, 3, 0>
10	15	1	124	129	< 97, 88, 4, 0, 3, 0>
11	20	1	119	143	< 97, 88, 4, 0, 3, 0>
12	25	1	119	156	< 97, 88, 4, 0, 5, 0>
13	30	1	119	156	< 97, 88, 4, 0, 5, 0>

Table 18 The genetic algorithm behavior in an environment with the growing workload

Iteration no.	C_R	C_{SLA}	No. of rented VMs	No. of SLA breaches	Optimal Configuration
1	1	1	127	1	< 95, 53, 0, 3, 0, 0>
2	1	5	127	1	< 95, 53, 0, 3, 0, 0>
3	1	10	153	0	< 80, 29, 0, 4, 0, 4>
4	1	15	153	0	< 80, 29, 0, 4, 0, 4>
5	1	20	153	0	< 80, 29, 0, 4, 0, 4>
6	1	25	153	0	< 80, 29, 0, 4, 0, 4>
7	1	30	153	0	< 80, 29, 0, 4, 0, 4>
8	5	1	96	80	< 96, 70, 4, 0, 3, 2>
9	10	1	96	80	< 96, 70, 4, 0, 3, 2>
10	15	1	96	80	< 96, 70, 4, 0, 3, 2>
11	20	1	90	91	< 97, 65, 4, 0, 4, 0>
12	25	1	88	94	< 99, 69, 4, 0, 4, 0>
13	30	1	88	94	< 99, 69, 4, 0, 4, 0>

As shown in the results, the proposed algorithm finds an optimum solution in accordance with the $\frac{C_{SLA}}{C_R}$ coefficient. When $\frac{C_{SLA}}{C_R}$ increases, the cloud client prefers to reduce the SLA violation cost (C_{SLA}) because the SLA cost becomes more expensive (see iterations 1 to 7 in Tables 17, 18, and 19). In this case, the algorithm decreases the upper and the lower thresholds which increases the spare capacity of the provisioned VMs (i.e. more VMs are added). This helps the cloud service to tolerate the unforeseen workload surges. Reducing the upper and the lower thresholds causes less SLA violations but increases the resource cost.

On the other hand, decreasing $\frac{C_{SLA}}{C_R}$ coefficient that is, the cloud client prefers to decrease the resource costs (see iterations 8 to 13 in Tables 17, 18, and 19) because the resource cost C_R is more expensive. This leads to configurations with higher thresholds, which accelerate the scaling actions and decreases the resource cost. Increasing the upper threshold causes the cloud service to fully use the provisioned VMs' capacities which makes the cloud service to become prone to more SLA violations.

The experimental results confirm that the proposed cost driven decision maker is able to find the configuration parameters that reduce the total cost based on the cloud clients' preferences.

Evaluation of the combined decision maker and the predictive auto-scaling system

In this section we put the two components (i.e., the self-adaptive predictor and the cost driven decision maker) together and evaluate the resulting predictive auto-scaling system against the Amazon auto-scaling system, a well-known and popular auto-scaling system in the commercial cloud environments [2]. The Amazon auto-scaling system's behavior is simulated and its auto-scaling cost is compared with the cost of our predictive auto-scaling system.

The Amazon auto-scaling policy uses the "*scaling based on the metrics*" technique to generate the scaling actions [21, 22] in which the cloud client defines the scaling policies. The Amazon policy scaling has similar parameters as our decision maker, i.e., *thrU, thrL, durU, durL, inU,* and *inL* (see eq. 4). In addition, Amazon lets clients decide the amount of resources that should be added to or removed from the underlying IaaS. Both Amazon auto-scaling system and our decision maker use the rule-based mechanism. Therefore, the same parameter values are used to configure both auto-scaling systems. Readers are encouraged to see [22] for more details about the Amazon scaling policies.

Table 19 The genetic algorithm behavior in an environment with the unpredictable workload

Iteration no.	C_R	C_{SLA}	No. of rented VMs	No. of SLA breaches	Optimal Configuration
1	1	1	155	2	< 96, 66, 0, 2, 2, 4>
2	1	5	155	2	< 96, 66, 0, 2, 2, 4>
3	1	10	153	1	< 96, 57, 0, 2, 0, 4>
4	1	15	158	0	< 92, 47, 0, 3, 0, 4>
5	1	20	158	0	< 92, 47, 0, 3, 0, 4>
6	1	25	158	0	< 92, 47, 0, 3, 0, 4>
7	1	30	158	0	< 92, 47, 0, 4, 0, 4>
8	5	1	121	106	< 97, 73, 4, 0, 4, 1>
9	10	1	112	128	< 97, 79, 4, 0, 4, 1>
10	15	1	105	188	< 97, 90, 4, 0, 3, 1>
11	20	1	96	201	< 99, 90, 4, 0, 3, 1>
12	25	1	96	201	< 99, 90, 4, 0, 3, 1>
13	30	1	96	201	< 99, 90, 4, 0, 3, 1>

In the simulation, three (periodic, growing, and unpredictable) different workload trace files are used to represent the three dominant workload patterns in the IaaS environment. According to the results in our previous work [14], 60% of the workload trace files are used to train the predictor component.

The same training trace files are used to identify an optimal set of the configuration parameters by the cost-driven genetic algorithm. Figures 8, 9, and 10 show the comparison results.

According to Fig. 8, the proposed auto-scaling system incurs less resource cost compared to the Amazon auto-scaling system. This means that the proposed system can forecast the future workload and generate more accurate decisions to reduce the resource cost.

In addition, the number of the SLA violations is compared in Fig. 9. Unlike the Amazon auto-scaling system, the proposed system in this paper uses the prediction methods to generate the scaling actions. This helps the proposed system to request the VMs ahead of time and prevents the under-provisioning condition. Therefore, the proposed system has less SLA violations compared to the Amazon auto-scaling system. Since the total cost is a metric used to evaluate the accuracy of the auto-scaling systems (see Section 2.4), it can be concluded that the proposed auto-scaling system is more accurate than the Amazon auto-scaling system. The total costs of the proposed auto-scaling system and the Amazon auto-scaling system are shown in Fig. 10 and the results show that the auto-scaling system reduces the total cost by:

12% in the case of periodic workload pattern,
25% for the growing workload pattern and,
10.6% with the unpredictable workload pattern.

Conclusions, open challenge, and future work

Deciding the optimal amount of resources in an IaaS cloud environment is very difficult and can be a double-edged sword either leading to over-provisioning or under-provisioning. Under provisioning can lead to SLA violation while over provisioning can cause wastage and excessive energy consumption. Auto-scaling systems are built to balance the cost-performance trade-off of over- or under-provisioning. Furthermore, neglecting the VM boot-up time and the difficulty associated with configuration parameters are the two main shortcomings of the rule-based auto-scaling systems. This paper investigates the impact of the VM boot-up time and the configuration parameters on the accuracy and cost of the rule-based auto-scaling systems, and proposes a predictive auto-scaling system that consists of a self-adaptive prediction suite [6] and a cost driven

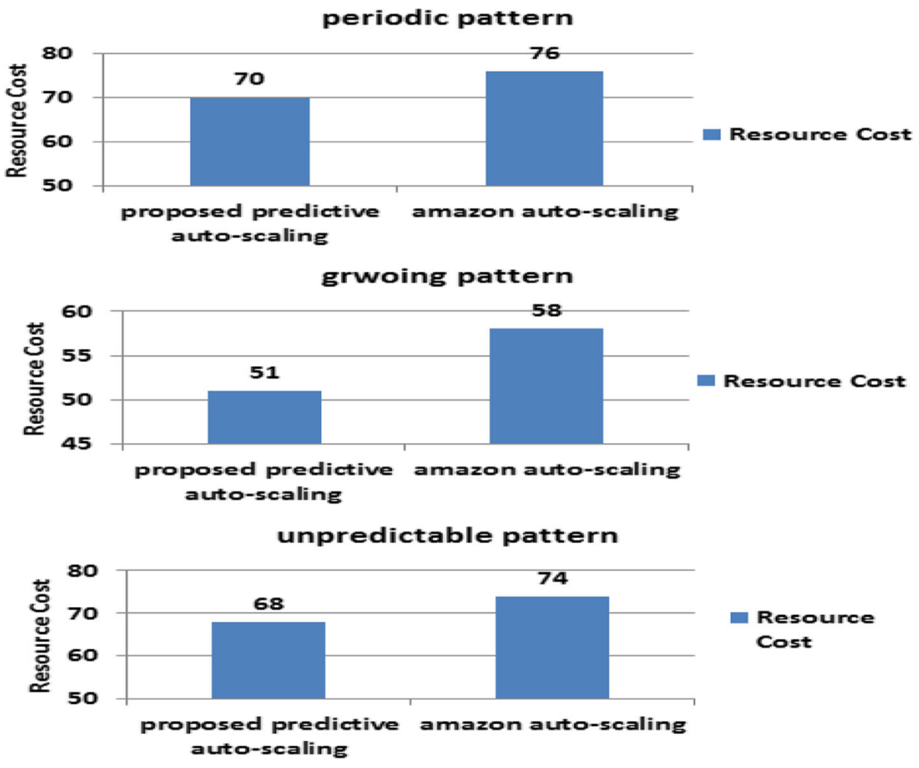

Fig. 8 Resource cost comparison

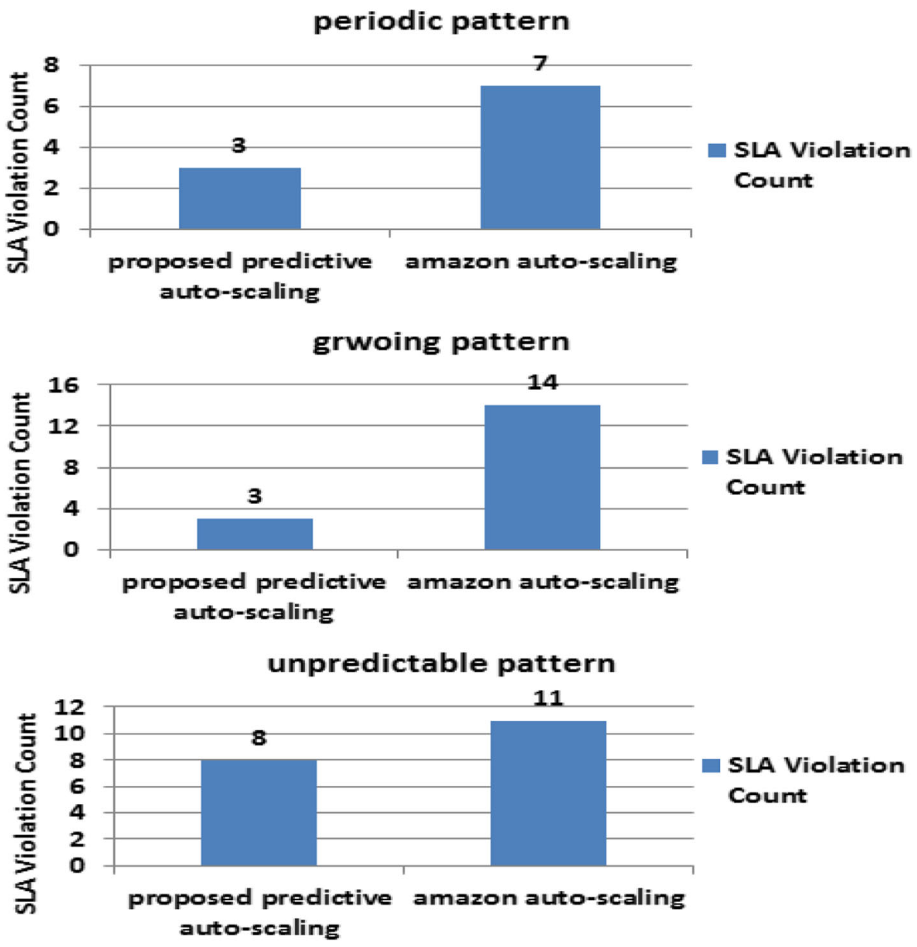

Fig. 9 SLA violation count comparison

decision maker. Our previous work [6] presents a self-adaptive prediction suite on which the prediction task of the proposed auto-scaling system is based. The proposed auto-scaling system bundles the prediction suite with a cost-driven decision maker to scale the business tier of the cloud services. In this paper we present the cost-driven decision maker component that uses a genetic algorithm to select optimum configuration parameters in a large search space for the auto-scaling system. According to the evaluation results, the proposed auto-scaling system can reduce the total auto-scaling cost by up to 25% compared with the Amazon auto-scaling system.

A threat to validity in this research work is that the proposed cost driven decision maker uses a genetic algorithm to find the optimal configuration of the rule-based decision makers. Changing the decision maker mechanism such as changing its parameter set or its scaling logic may change the genetic algorithm results.

Some consideration for future work includes the following. The proposed auto-scaling system uses the predictive approach to generate the scaling actions ahead of time to prevent the under-provisioning and the over-provisioning conditions. Another approach may be to replace the VMs with the container-based virtualization (such as Docker containers [32]). Using the containers instead of the VMs may reduce the probability of the SLA violations. Therefore, investigating the impact of the container-based virtualization on the accuracy of the auto-scaling systems can be considered as a future work.

Another future work is to be able to stream the workload data. Streaming data analysis in real-time mode is very important to obtain useful knowledge from the data and to see what is happening now. This will allow the user of the decision maker model to react quickly to the predictions as well as the optimal configurations.

In addition, the proposed system is limited to the business tier auto-scaling. Extending the auto-scaling system

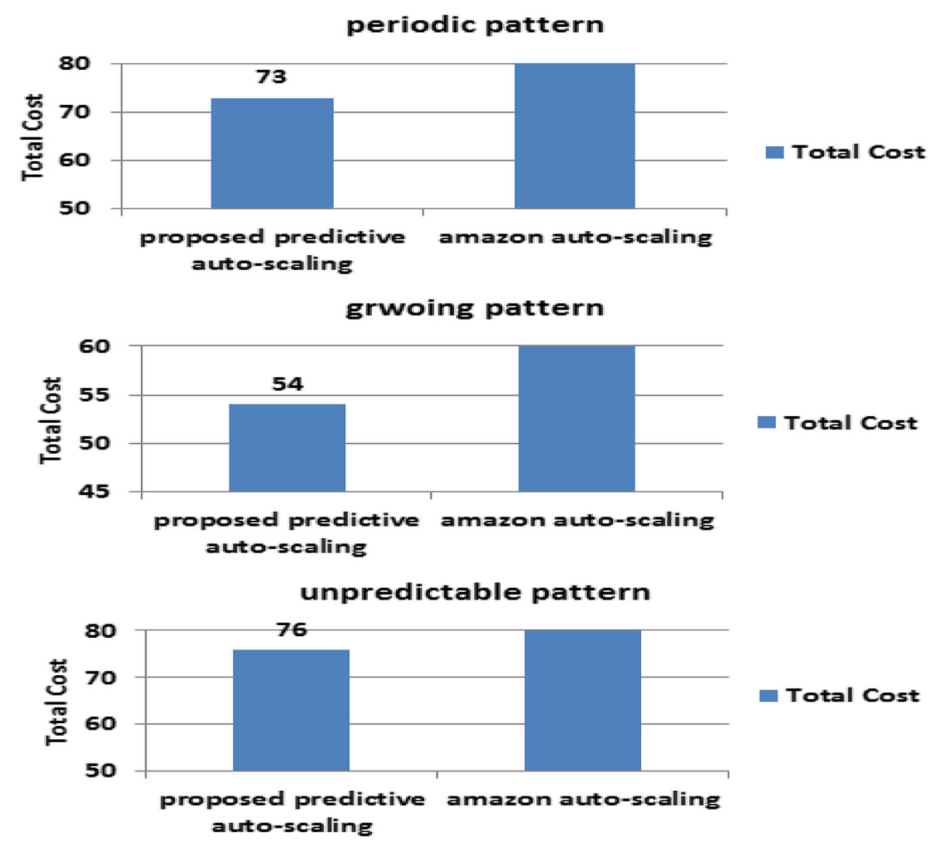

Fig. 10 Total cost comparison

to manage the database tier can be investigated in the future studies [33]. The logic of the database tier and the business tier auto-scaling are similar. However, unlike the business tier, the database tier includes a set of *data-files* which can greatly affect the auto-scaling strategy.

Finally, the proposed system assumes there is only one VM type in the IaaS environment. In future studies, the proposed genetic algorithm can be improved to configure the decision makers with several VM types. Considering heterogeneous VM types converts the auto-scaling problem to become a scheduling problem, which can also] be solved using genetic algorithms.

Abbreviations
SLA: Service level agreement; SLAs: Service level agreements; VM: Virtual machine; VMs: Virtual machines; QoS: Quality-of-service; MLP: Multi-layer perception; MLPWD: Multi-layer perception with weight decay; SVM: Support vector machine; thrU: Upper threshold; thrL: Lower threshold; durU: Upper scaling duration; durL: Lower scaling duration; inU: Upper cool-down duration; inL: Lower cool-down duration; CR($): Resource cost; C_b: Cost factor; C_{vm}: VM hourly rate; λ: Incoming workload prediction; D: Average service demand; $Cap_{ceiling}$: Ceiling capacity; $CapU_{vm}$: Upper threshold of the number of the requests that can be handled by each of the VMs per second; QNM: Queuing network model

Acknowledgements
Not Applicable.

Funding
This research is partly sponsored by NSERC (Natural Science and Engineering Council of Canada) Discovery and Engage grants.

Availability of data and materials
Not applicable.

Authors' contributions
The authors equally contributed to this research and the paper. All authors read and approved the final manuscript.

Authors' information
Ali Yadavar Nikravesh received the B.S. and the M.S. degrees in Software Engineering from Shahid Beheshti University, Iran and the Ph.D. degree in Computer Engineering from Carleton University, Canada. In September 2016 he joined Microsoft Corporation, Seattle, USA where he is now a Software Engineer. His research interests include: Cloud computing, machine learning algorithms, and time-series prediction.
Samuel A. Ajila is currently an associate professor of engineering at the Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada. He received B.Sc. (Hons.) degree in Computer Science from University of Ibadan, Ibadan, Nigeria and M.Sc., DEA, and Ph.D. degrees in Computer Engineering specializing in Software Engineering and Knowledge-based Systems from LORIA, Université Henri Poincaré – Nancy I, Nancy, France. His research interests are in the fields of Software Engineering, Cloud Computing, Big Data Analytics and Technology Management.
Chung-Horng Lung received the B.S. degree in Computer Science and Engineering from Chung-Yuan Christian University, Taiwan and the M.S. and Ph.D. degrees in Computer Science and Engineering from Arizona State University. He was with Nortel Networks from 1995 to 2001. In September 2001, he joined the Department of Systems and Computer Engineering,

Carleton University, Ottawa, Canada, where he is now a Professor. His research interests include: Software Engineering, Cloud Computing, and Communication Network.

Competing interests
To the best of the authors' knowledge, there is no competing interest whatsoever and so far as this research and the content of the paper is concerned.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 28 May 2018 Accepted: 26 October 2018
Published online: 07 November 2018

References

1. A. A. Bankole, "Cloud client prediction models for cloud Resrouce provisioning in a multitier web application environment," MASc thesis report, Carleton University, 2013
2. Bankole AA, Ajila SA (2013) Cloud client prediction models for cloud resource provisioning in a multitier web application environment," 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, pp 156–161
3. Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. J Grid Comput 12(4):559–592
4. Beloglazov A, Buyya R (2011) Adaptive threshold-based approach for energy-efficient consolidation of virtual Machines in Cloud Data Centers," Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science, vol 2010, p 6
5. Islam S, Keung J, Lee K, Liu A (2012) Empirical prediction models for adaptive resource provisioning in the cloud. Futur Gener Comput Syst 28(1):155–162
6. Nikravesh AY, Ajila SA, Lung C-H (2017) A self-adaptive prediction suite for the cloud resource provisioning. Journal of Cloud OCmputing 6:4
7. Amazon Elastic Compute Cloud (Amazon EC2), 2013. [Online], Available: http://aws.amazon.com/ec2
8. RackSpace, The Open Cloud Company, 2012. [Online], Available: http://rackspace.com
9. RightScale Cloud management, 2012. [Online], Available: http://rightscale.com
10. Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows," Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. - SC '11, p 1
11. Benediktsson JA, Kanellopoulos I (1999) Classification of multisource and hyperspectral data based on decision fusion. IEEE Trans. Geosci. Remote Sens 37(3):1367–1377
12. Hasan MZ, Magana E, Clemm a, Tucker L, Gudreddi SLD (2012) Integrated and autonomic cloud resource scaling," 2012 IEEE Network Operations and Management Symposium, pp 1327–1334
13. "Carleton Auto-Scaling Simulator," 2016. [Online]. Available: https://github.com/alinikravesh/Carleton-Auto-Scaling-Simulator/. [Accessed: 01-Jan-2016]
14. Nikravesh AY, Ajila SA, Lung C-H (2015) Evaluating sensitivity of auto-scaling decisions in an environment with different workload patterns," IEEE 39th Annual Computer Software Applications Conference, pp 415–420
15. Biswas A, Majumdar S, Nandy B, El-Haraki A (2015) Predictive auto-scaling techniques for clouds subjected to requests with service level agreements," 2015 IEEE World Congress on Services, pp 311–318
16. Nikravesh AY, Ajila SA, Lung C-H (2014) Measuring prediction sensitivity of a cloud auto-scaling system," in Proceedings - 38th IEEE annual international computers, software and applications conference workshop, pp 690–695
17. A. Y. Nikravesh, S. A Ajila, and C.-H. Lung, "Towards an autonomic auto-scaling prediction system for cloud resource provisioning," in SEAMS, 2015
18. Lazowska E, Zahorjan J, Graham S, Sevcik K (1984) Quantitative system performance, computer system analysis using queueing network models. Prentice-Hall Inc, New Jersey
19. Casalicchio E, Silvestri L (2013) Autonomic Management of Cloud-Based Systems: the service provider perspective. In: Computer and Infiormation sciences III, pp 487–494
20. Menasce D, Dowdy L, Almeida V (2004) Performance by Design: Computer Capacity Planning By Example, 1st edn. Prentice Hall, New Jersey
21. Xiao Z, Chen Q, Luo H (2014) Automatic scaling of internet applications for cloud computing services. IEEE Trans Comput 63(5):1111–1123
22. "Amazon Scaling Based on Metric," 2016. [Online]. Available: http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html#policy-updating-console
23. Nikravesh AY, Ajila SA, Lung C-H (2014) Cloud resource auto-scaling system based on hidden Markov model (HMM)," *2014 IEEE International Conference on Semantic Computing*, pp 124–127
24. "Layered Queueing Network Solver Software Package." [Online]. Available: http://www.sce.carleton.ca/rads/lqns/. [Accessed: 27-Apr-2016]
25. Schmid M, Thoss M, Termin T, Kroeger R (2007) A generic application-oriented performance instrumentation for multi-tier environments. In: 10th IFIP/IEEE international symposium on integrated network management 2007, IM '07, pp 304–313
26. Franks G, Petriu D, Woodside M, Xu J, Tregunno P (2006) Layered bottlenecks and their mitigation," Third International Conference on the Quantitative Evaluation of Systems, QEST, pp 103–112
27. Yu J, Buyya R (2006) Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci Program 14:217–230
28. "Introduction to Genetic Algorithms" [Online]. Available: http://www.obitko.com/tutorials/genetic-algorithms/about.php
29. Safe M, Carballido J, Ponzoni I, Brignole N (2004) On stopping criteria for genetic algorithms," Proceedings of 17th Brazilian Symposium on Advanced Artificial Intelligence, pp 405–413
30. Oliveto PS, Witt C (2015) Improved time complexity analysis of the simple genetic algorithm. Theor Comput Sci 605:21–41. https://doi.org/10.1016/j.tcs.2015.01.002
31. Bart Rylander, Computational Complexity and the Genetic Algorithm, A Dissertation Presented in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy, College of Graduate Studies, University of Idaho, 2001
32. "Docker," 2016. [Online]. Available: https://www.docker.com/
33. Urgaonkar B, Shenoy P, Chandra A, Goyal P (2005) Agile dynamic provisioning of multi-tier internet applications. ACM Transactions on Autonomous and Adaptive Systems 2005(1):217–228