PREDICT EFFICIENT CLOUD COST-DRIVEN DECISION MAKER USING GENETIC ALGORITHMS BASED AUTO-SCALING SYSTEMS

[1*]R CHANDRASHEKAR, A PADMA[2], N SHILPA[3]

[123]ASSISTANT PROFESSOR, DEPT OF CSE, SRI INDU INSTITUTE OF ENGINEERING AND TECHNOLOGY, IBRAHIMPATNAM,SHERIGUDA, TELANGANA

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).

I. 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.

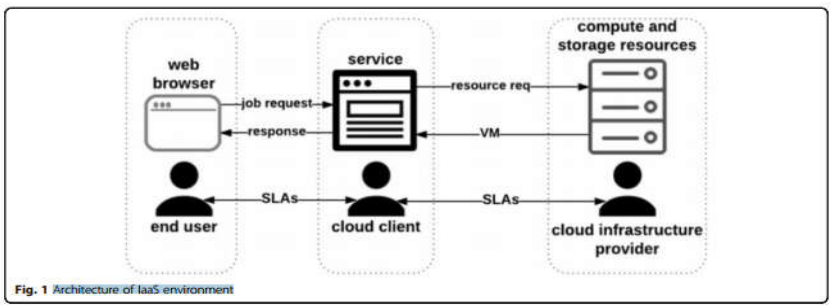

Fig. 1 Architecture of IaaS environment

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
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 sys-tem 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 re-source 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 pro-posed genetic algorithm considers the cloud client's cost preferences to find the optimum configuration set.

II. Related work

2.1 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 is suitable when the changes in the workload are known

ahead of time [10]. However, not all of the cloud services have time-based workload pat-terns, 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 pre-diction 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.

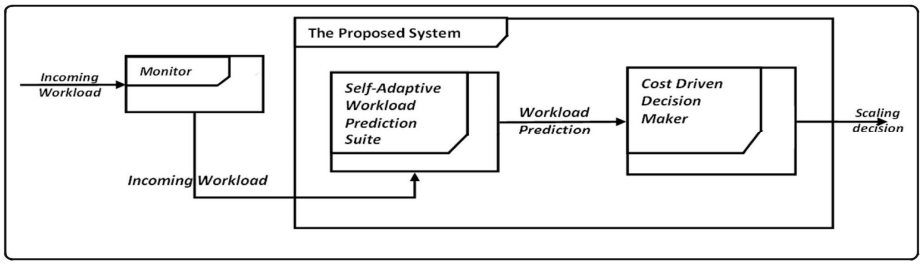

Fig. 2 Predictive Auto-scaling system architecture

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 do-main [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.

2.2 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 pat-tern. Therefore, the self-adaptive suite automatically chooses:

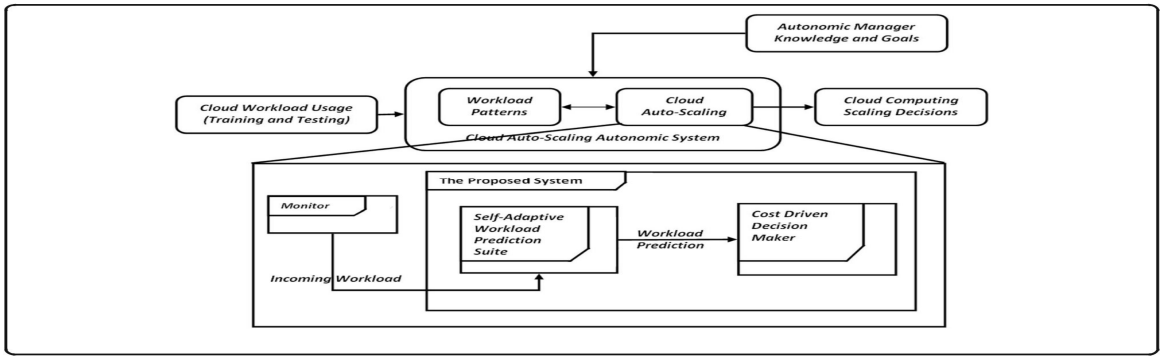

Fig. 3 Projection of the cloud auto-scaling autonomic element

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 work-load 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
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] gene-rates 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.

2.3 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 (CR) and SLA violation cost (CSLA).

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:

$$C_R = \sum_{t=0}^{T} n_t \times c_{vm}$$

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:

$$v_t = \sum_{req=1}^{N} v_{t,req}$$

$$v_{t,req} = \begin{cases} 1 & if\ (r_{req} - R) > 0 \\ 0 & otherwise \end{cases}$$

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:

$$C_{SLA} = \sum_{t=o}^{T} v_t \times c_b$$

III. Proposal work

3.1 Genetic Algorithm (GA)

The genetic algorithm is a random-based classical evolutionary algorithm. By random here we mean that in order to find a solution using the GA, random changes applied to the current solutions to generate new ones. Note that GA may be called Simple GA (SGA) due to its simplicity compared to other EAs.

GA is based on Darwin's theory of evolution. It is a slow gradual process that works by making changes to the making slight and slow changes. Also, GA makes slight changes to its solutions slowly until getting the best solution.

Here is the description of how the GA works:

GA works on a population consisting of some solutions where the population size (popsize) is the number of solutions. Each solution is called individual. Each individual solution has a chromosome. The chromosome is represented as a set of parameters (features) that defines the individual. Each chromosome has a set of genes. Each gene is represented by somehow such as being represented as a string of 0s and 1s as in the next diagram.

Also, each individual has a fitness value. To select the best individuals, a fitness function is used. The result of the fitness function is the fitness value representing the quality of the solution. The higher the fitness value the higher the quality the solution. Selection of the best individuals based on their quality is applied to generate what is called a mating pool where the higher quality individual has higher probability of being selected in the mating pool.

The individuals in the mating pool are called parents. Every two parents selected from the mating pool will generate two offspring (children). By just mating high-quality individuals, it is expected to get a better quality offspring than its parents. This will kill the bad individuals from generating more bad individuals. By keeping selecting and mating

high-quality individuals, there will be higher chances to just keep good properties of the individuals and leave out bad ones. Finally, this will end up with the desired optimal or acceptable solution.

But the offspring currently generated using the selected parents just has the characteristics of its parents and no more without changes. There is no new added to it and thus the same drawbacks in its parents will actually exist in the new offspring. To overcome such problem, some changes will be applied to each offspring to create new individuals. The set of all newly generated individuals will be the new population that replaces the previously used old population. Each population created is called a generation. The process of replacing the old population by the new one is called replacement. The following diagram summarizes the steps of GA.

There are two questions to be answered to get the full idea about GA:

1. How the two offspring are generated from the two parents?
2. How each offspring gets slightly changed to be an individual?

We will answer these questions later.

3.2 Chromosome Representation and Evaluation

There are different representations available for the chromosome and the selection of the proper representation is problem specific. The good representation is what makes the search space smaller and thus easier search.

The representations available for the chromosome including:

- Binary: Each chromosome is represented as a string of zeros and ones.
- Permutation: Useful for ordering problems such as travelling salesman problem.
- Value: The actual value is encoded as it is.

For example, if we are to encode the number 7 in binary, it might look as follows:

Each part of the above chromosome is called gene. Each gene has two properties. The first one is its value (allele) and the second one is the location (locus) within the chromosome which is the number above its value.

Each chromosome has two representations.

1. **genotype: The set of genes representing the chromosome.**
2. **phenotype:** The actual physical representation of the chromosome.

In the above example, binary of 0111 is the genotype and 7 is the phenotype representation.

After representing each chromosome the right way to serve to search the space, next is to calculate the fitness value of each individual. Assume that the fitness function used in our example is:

f(x) = 2x+2 Where x is the chromosome value

Then the fitness value of the previous chromosome is:

f(7) = 2(7)+2=16

The process of calculating the fitness value of a chromosome is called evaluation.

Initialization

After getting how to represent each individual, next is to initialize the population by selecting the proper number of individuals within it.

Selection

Next is to select a number of individuals from the population in the mating pool. Based on the previously calculated fitness value, the best individuals based on a threshold are selected. After that step, we will end selecting a subset of the population in the mating pool.

3.3 Variation Operators

Based on the selected individuals in the mating pool, parents are selected for mating. The selection of each two parents may be by selecting parents sequentially (1-2, 3-4, and so on). Another way is random selection of the parents.

For every two parents selected, there are a number of variation operators to get applied such as:

1. **Crossover (recombination)**
2. **Mutation**

The next diagram gives an example for these operators Crossover

Crossover in GA generates new generation the same as natural mutation. By mutating the old generation parents, the new generation offspring comes by carrying genes from both parents. The amount of genes carried from each parent

is random. Remember that GA is random-based EA. Sometimes the offspring takes half of its genes from one parent and the other half from the other parent and sometimes such percent changes. For every two parents, crossover takes place by selecting a random point in the chromosome and exchanging genes before and after such point from its parents. The resulting chromosomes are offspring. Thus operator is called single-point crossover.

Mutation

Next variation operator is mutation. For each offspring, select some genes and change its value. Mutation varies based on the chromosome representation but it is up to you to decide how to apply mutation. If the encoding is binary (i.e. the value space of each gene have just two values 0 and 1), then flip the bit value of one or more genes. But if the gene value comes from a space of more than two values such as 1,2,3,4, and 5, then the binary mutation will not be applicable and we should find another way. One way is by selecting a random value from such set of values as in the next diagram.

3.4 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 con-figure 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 .

The objective here is to locate the best value for each of the design parameters with the end goal that the configured rule-based decision maker minimizes the last auto-scaling cost. Since the space of substantial values for each of the parameters is known, the universal set of possible arrangements can be created where each arrangement is a legitimate blend of the parameters. Then to locate the ideal arrangement, the search space (i.e., the universal set of the arrangements) is traversed and the arrangement with the least auto-scaling cost is found. To measure the auto-scaling cost of a given arrangement, a decision maker is configured with the parameters of that arrangement, and an auto-scaling recreation is kept running for a predefined length to calculate the aggregate auto-scaling cost of the decision maker. In this paper, an in-house reproduction package [13] is implemented and used to do the recreations.

Effect of VM-boot-up time, SMARTKILL and setup parameters on AUTOSCALING exactness and expenses

In this section, we present the result of three experiments on the effect of VM boot-up time on auto-scaling exactness, and the effect of the keen kill technique on auto-scaling cost factors and the effect of design parameters on auto-scaling precision. The results of the effect of setup parameters demonstrate how troublesome it is for a cloud client to handle the design parameters. What's more, the savvy slaughter results demonstrate how essential the keen murder is in decreasing resource cost or reducing SLA infringement.

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

An in-house recreation package [13] is developed and used to complete reenactment on the effect of VM boot-up time on the cost elements of a rule-based decision maker.

Effect of the arrangement parameters on the auto-scaling precision

This section investigates the effect of the arrangement 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 calculation (or some other calculation) to reduce the space search time as well as to discover ideal setup for the different parameters. Note that our rule-based decision makers have six design parameters: thrU, thrL, durU, durL, inU, and inL.

The upper threshold (thrU)

The primary iteration analyses the influence of the thrU on the SLA infringement and the resource cost. The values of the other design parameters (except the thrU) are held steady amid the reenactment to isolate the relationship between the thrU and the cost factors.

The lower threshold (thrL)

The second iteration of the experiment investigates the effect of the thrL on the SLA infringement and the resource cost.

The freezing lengths (inU and inL)

The last iteration of the experiment studies the effects of the inU and the inL parameters on the cost factors.

3.6 Finding an optimum configuration for AUTOSCALING problem

The genetic calculation (GA) applies the evolution principle to provide a vigorous search technique that finds an amazing arrangement in a large search space in polynomial time. A genetic calculation combines the exploitation of the best arrangements from the past searches with the exploration of the new regions of the arrangement space [27]. Any arrangement in the search space is represented by a chromosome.

A genetic calculation keeps up a populace of the chromosomes that evolves over the generations (i.e. iterations). The nature of a chromosome in the populace is determined by a fitness work. The fitness value indicates how great a chromosome is compared to the other chromosomes in the populace [27]. A run of the mill genetic calculation has the accompanying steps:

1. Create an underlying populace comprising of arbitrarily generated arrangements.
2. Generate a new posterity by applying the genetic operators which are: selection, crossover, and transformation, one after the other.
3. Calculate the fitness value of the chromosomes.
4. Repeat steps 2 and 3 until the point when the calculation converges to an ideal arrangement.

So as to use the genetic calculation principle to solve the auto-scaling problem, the representation of the chromosomes in the populace, the fitness work, and the genetic operators ought to be determined.

3.7 Chromosome representation

In the rule-based systems, a feasible arrangement is required to meet the accompanying conditions:

The upper threshold (thrU) ought to be less than or equal to the upper furthest reaches of the performance marker. For instance, if CPU usage is the performance marker, the upper threshold can't be greater than 100%.

The lower threshold (thrL) ought to be greater than or equal to the lower furthest reaches of the performance marker. For instance, if CPU usage is the performance marker, the lower threshold can't 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., $thrU > thrL$).

Complexities of Algorithms 1 and 2

Change operator introduces the irregular alterations in the genes of the chromosomes. The purpose of the transformation operator is to keep up the diversity in the populace and stay away from a premature convergence. For the value encoding, the change operator is implemented by adding a modest number to the arbitrarily selected genes. In the auto-scaling problem area, one of the genes (i.e., an arrangement parameter) is arbitrarily picked and its value is altered.

Finding the ideal values for the genetic calculation.

The precision of the rule-based calculation is defined by the auto-scaling cost which is related to the set of the setup parameters that is found by the genetic calculation.

Populace size

The populace size indicates the number of arrangements that are examined in each of the iterations of the genetic calculation. The populace ought to be huge enough to cover the diversity of the search space. Then again, the larger populace sizes increase the length of the experiment. In the main iteration of the experiment, the relationship between the populace size and the exactness of the genetic calculation is investigated.

Stop condition

There are three termination conditions that are used in the genetic calculations: (1) when an upper limit on the number of the generations is reached, or (2) when an upper limit on the number of evaluations of the fitness work is reached, or (3) when the chance of achieving a noteworthy change in the next generations is excessively low [29].

Crossover rate

The crossover rate or the crossover likelihood indicates a proportion of what number of couples will be selected to generate a new posterity.

Change rate

the change operator relates to the exploration capacity of the genetic calculation. While the crossover operator tries to converge to a specific point in the search space, the transformation operator keeps away from the convergence and explores more areas.

Conclusions

Deciding the ideal measure of resources in an IaaS cloud environment is very troublesome and can be a double-edged sword either leading to over-provisioning or under-provisioning. Under-provisioning can lead to SLA infringement while over provisioning can cause wastage and excessive energy utilization. Auto-scaling systems are worked to balance the cost-performance trade-off of over-or under-provisioning. Furthermore, neglecting the VM boot-up time and the trouble associated with arrangement parameters are the two principle deficiencies of the rule-based auto-scaling systems. This paper investigates the effect of the VM boot-up time and the setup parameters on the precision and cost of the rule-based auto-scaling systems and proposes. A predictive auto-scaling system that comprises of a self-adaptive prediction suite [6] and a cost-driven decision maker. Our previous work [6] presents a self-adaptive prediction suite on which the prediction undertaking of the proposed auto-scaling system is based. The proposed auto-scaling system bundles the pre-phrasing 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 calculation to select ideal arrangement parameters in a large search space for the auto-scaling system.

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.

AUTHOR DETAILS

Mr. R CHANDRASHEKARHAS BEEN AN ASSISTANT PROFESS(IN THE DEPARTMENT OF computer science and engineering, SRI INDU INSTITUTE OF ENGINEERING AND TECHNOLOGY, IBRAHIMPATNAM(M),SHERIGUDA (VILL), RANGAREDDY (TELANGANA. HE TEACHES VARIOUS UG COURCSE SUBJE IN THE DEPT CSE. HER RESEARCH INTERESTS INCLUDE DA MINIG,CLOUD COMPUTING,BIG DATA .

Mrs. A PADMA HAS BEEN AN ASSISTANT PROFESSOR IN THE DEPARTMENT OF computer science and engineering, SRI INDU INSTITUTE OF ENGINEERING AND TECHNOLOGY, IBRAHIMPATNAM(M),SHERIGUDA (VILL), RANGAREDDY (D.T), TELANGANA. SHE TEACHES VARIOUS UG COURCSE SUBJECTS IN THE DEPT CSE. HER RESEARCH INTERESTS INCLUDE MOBILE COMPUTING, DATA MINIG, CLOUD COMPUTING, COMPUTER NETWORKS.

Mrs. N SHILPA HAS BEEN AN ASSISTANT PROFESSOR IN THE DEPARTMENT OF computer science and engineering, SRI INDU INSTITUTE OF ENGINEERING AND TECHNOLOGY, IBRAHIMPATNAM(M), SHERIGUDA (VILL), RANGAREDDY (D.T), TELANGANA. SHE TEACHES VARIOUS UG COURCSE SUBJECTS IN THE DEPT CSE. HER RESEARCH INTERESTS INCLUDE DATA MINIG,CLOUD COMPUTING,BIG DATA