Automatic Resource Scaling for Web Applications in the Cloud

Ching-Chi Lin[1,2], Jan-Jan Wu[1], Pangfeng Liu[2], Jeng-An Lin[2], and Li-Chung Song[2]

[1] Institute of Information Science Research Center for Information Technology Innovation
Academia Sinica, Taipei, Taiwan
{deathsimon,wuj}@iis.sinica.edu.tw
[2] Department of Computer Science and Information Engineering Graduate
Institute of Networking and Multimedia National Taiwan University, Taipei, Taiwan
{pangfeng,r99944038,r00922089}@csie.ntu.edu.tw

Abstract. Web applications play a major role in various enterprise and cloud services. With the popularity of social networks and with the speed at which information can be disseminate around the globe, online systems need to face ever-growing, unpredictable peak load events.

Auto-scaling technique provides on-demand resources according to workload in cloud computing system. However, most of the existing solutions are subject to some of the following constraints: (1) replying on user provided scaling metrics and threshold values, (2) employing the simple Majority Vote scaling algorithm, which is ineffective for scaling Web applications, and (3) lack of capability for predicting workload changes. In this work, we propose an effective auto-scaling strategy, called *Work-load Based* scaling algorithm, for Web applications. Our proposed scaling strategy is not subject to the aforementioned constraints, and can respond to fluctuated workload and sudden workload change in a short time without relying on over-provisioning of resources. We also propose a new method for analyzing the trend of workload changes. This trend analysis method provides useful information to the scaling algorithm to avoid unnecessary scaling actions, which in turn shortens the response time of requests. The experiment results show that the hybrid *Workload Based* and *trend analysis* method keeps response time within 2 seconds even when facing sudden workload change.

Keywords: Cloud Computing, Auto-Scaling, Web Applications, Resource Provisioning, Trend Analysis.

1 Introduction

Web applications play a major role in various enterprise and cloud services. Many Web applications, such as eBanking, eCommerce and online gaming, face fluctuating loads. Some of the loads are predictable, such as the workload around a holiday for eShopping services. However, with the popularity of social networks and with the speed at which information can disseminate around the globe, online systems need to face ever-growing, unpredictable peak load events.

J.J. Park et al. (Eds.): GPC 2013, LNCS 7861, pp. 81–90, 2013.

Auto-scaling is a solution that not only maintains application service quality but also reduces wasted resources while facing fluctuating loads. The basic idea of *auto-scaling* is to estimate the load for short window of time, and then be able to up-scale or down-scale the resources when there is a need for it. Many cloud services, such as Amazon EC2 [1] and Google App Engine [2], have proposed auto-scaling service. Other software such as Scalr [3] and RightScale [4] provide auto-scaling mechanism that can apply to cloud environments. However, most of the existing solutions are subject to some of the following constraints: (1) replying on user-provided scaling metrics and threshold values, (2) employing the simple *Majority Vote* scaling algorithm, which we will show in this paper to be ineffective for scaling Web applications, and (3) lack of capability for predicting workload change, and thus may result in unnecessary scaling actions.

We develop an auto-scaling system, *WebScale*, which is not subject to the aforementioned constraints. *WebScale* monitors the behavior of the applications and the system, and based on the collected metrics, decides in real time whether the number of VMs for an application needs to be increased or decreased. Because of page limit, in this paper, we focus on the new algorithms we propose for scaling resources for Web applications.

The main contributions of this work are as follows. (1) We show that the incoming requests from the clients (instead of standard metrics) and the HTTP response time can characterize the workload/performance behavior of Web applications more accurately. Based on this observation, we devise an effective scaling algorithm called *Workload-Based* algorithm for Web applications. (2) We propose an algorithm to analyze the trend of workload change in a Web application. This trend analysis algorithm can significantly reduce the number of peaks (longer than 2 seconds) in response time caused by workload fluctuating. (3) Our experiment results with workload generated by *httperf* [5] demonstrate that our scaling strategy can keep the average response time of Web applications within 2 seconds even when facing sudden load change.

The rest of the paper is organized as follows. Section 2 presents the Workload-Based scaling algorithm and the Trend Analysis algorithm. Section 3 presents and analyzes experiment results. Section 4 describes related work. Finally, Section 5 gives some concluding remarks.

2 Scaling Algorithms

In this section, we first give a brief overview of the widely used scaling algorithm, *Majority Vote*. We then present our *Workload-Based* scaling algorithm. Finally, we present our *Trend Analysis* technique, which co-works with the scaling algorithm to achieve better performance.

2.1 Overview of Majority Vote

Majority Vote selects the choice with the most properties among all choices. There are three choices, **scale in**, **scale out**, and **no scale**, for a VM when making decision.

Each VM makes their choice according to their current loading, and a final scaling decision will be made by majority vote.

Each VM makes its choice according to a chosen metric, such as CPU load or memory usage. For each chosen metric, there are two thresholds, *threshold_H* and *threshold_L,* which represent the high threshold and low threshold respectively. If the chosen metric of a VM is greater than *threshold_H*, the VM will choose **scale out**. On the other hand, if the chosen metric is smaller than *threshold_L*, the VM will choose **scale in**. Otherwise, the choice will be **no scale**.

2.2 Workload-Based Algorithm

The *Workload-Based* Algorithm determines the number of running VMs needed for the business-logic tier and the number of database servers needed for the data-access tier, based on the incoming workload. The latter is the number of requests per second for the business-logic tier, and the number of SQL queries per second for the data-access tier. In real world web applications, requests sent by clients will be received by the front-end load balancer, and then distributed to the back-end VMs. Since each VM has a capacity limitation, e.g. maximum number of requests per second a running VM can handle simultaneously, we can calculate the number of VMs needed to process the current workload, and make scaling decisions based on the number of current running VMs. The same applies to the number of database servers.

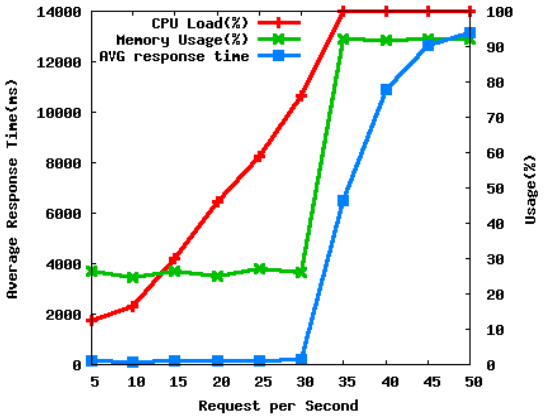

Fig. 1. Relationship between requests per second, average response time, CPU load, and memory usage

The reason that we choose requests per second instead of standard metrics to characterize workload of Web applications is that standard metrics fail to indicate "how busy" a VM is. Figure 1 shows the relationship between requests per second, two of the standard metrics, and the average response time. The average response time dramatically increases if the workload is over 30 requests per second. However, the standard metrics, CPU load and memory usage, in this example, remains constant when the number of requests per second is larger than 35. The standard metrics fail to

characterize the workload, thus cannot provide accurate information to decide the number of VMs needed. On the other hand, using requests per second as the metric can avoid this problem. With this metric, we can calculate the number of new running VMs needed to make the average response time decrease to an acceptable range.

The assumption that a VM has a capacity limitation; that is, it can only process a fixed number of requests per second simultaneously, is reasonable because of the need to maintain QoS. It has been shown in many previous works [6, 7, 8] that the types of requests to a Web application are bounded by a small constant, and that the percentage of each kind of requests to a Web application can be estimated. With such information, we can determine the capacity limitation of a VM.

We propose a variation of *parameter selection* scheme [9] to determine the capacity limitation of a VM. Our parameter selection scheme collects the system and performance information of a VM under different workload. This information is stored in a list, sorted by workload in ascending order. We can choose the maximum workload value from the list such that the corresponding performance satisfies the QoS requirement. This workload value is used as the capacity limitation of the VM. The following is an example on how to decide the capacity limitation of a VM or database server.

Table 1. Average response time(ms) under different workload combination

req/s	100% Q1	75% Q1 + 25% Q2	50% Q1 + 50% Q2	100% Q2
15	140.6	237.6	326.7	503.5
20	159.0	251.4	367.2	18714.5
25	157.7	6964.0	14082.4	22805.6
30	8222.3	13860.2	17525.5	22552.0

We use Table 1 to illustrate how we determine the capacity limitation of a VM using the parameter selection scheme. We use MediaWiki [10] as the web application. We assume that there are two kinds of requests, Q1 and Q2. Q1 requests a static Wiki page and Q2 requests a dynamic page which lists the links of top 100 articles in the database, sorted in descending order. Q2 has longer processing time than Q1. This table shows that the average response time dramatically increases if the request per second grows beyond a certain value under different workload combinations. Furthermore, the capacity limitation decreases when the percentage of more expensive requests (i.e., Q2 in this example) increases.

$$S = \frac{W}{C} - R \tag{1}$$

Our *Workload-Based* algorithm works as follows. For every fixed time interval, or "monitor interval", our WebScale scaling system collects the current workload information. By dividing the current workload W by the VM capacity C, we have the number of VMs needed. Then we subtract the current number of running VM R from this number, and get S, the amount of VM to be scaled. If S is positive, the decision will be **scale out**; on the other hand, if S is negative, the decision will be **scale in**. The same strategy applies to the scaling of database servers.

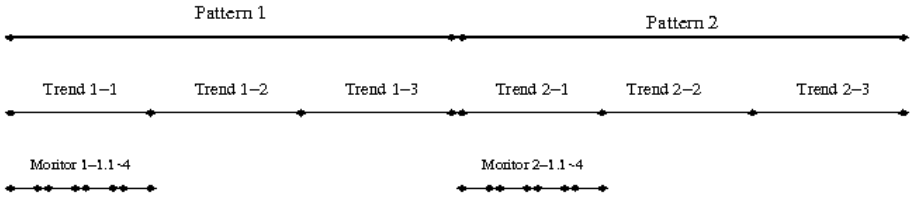

Fig. 2. Relation between different intervals

2.3 Trend Analysis

Some workloads exhibit periodic behaviors, e.g. stock market, enterprise applications, which we can take advantages while making scaling decisions. Periodic behavior means that similar behavior of the workload shows up every fixed length of time, thus we can predict the coming workload by historical data. Workload prediction has been studied by some previous works [11, 12]. Several different strategies have been proposed to predict application workload.

Instead of accurately predicting the workload value, for auto-scaling, it suffices to only predict the *trend* of workload change. *Trend* is the **direction** of workload changing in a fixed size of time, or "trend interval". There are three possibilities, *up*, *down*, or *constant*. The workload *trend* of an application can be acquired from historical workload data. For most web applications that exhibit periodic behaviors, the pattern length is usually one day or one week. Given the pattern length, we can divide the historical data into pieces, each with length equals to the pattern length, and determine the trend of the pattern.

A pattern consists of several trend intervals, which can be further divided into monitor intervals. Figure 2 shows the relationship between the pattern of workload, trend interval, and monitor interval. Scaling decisions are made in every monitor interval and compared with the trend of the trend interval from previous pattern. For example, monitor interval 2-1.2 makes a decision "scale out". This decision is then compared with the trend of trend interval 1-2 for confliction. Trend 2-1 will be updated after all the monitor intervals (2-1.1˜4) make their decisions.

The trend analysis technique works as a helper to the scaling algorithms by providing workload trend information to the scaling algorithms to make more "correct" decision while handling workloads with periodic behaviors. If a *scale in* decision "conflicts" with the trend, i.e., the decision is *scale in* while the trend is *scale out*, then the decision will be canceled and *no scale* will be the new decision. The rationale is to avoid removing VMs during workload increasing.

3 Experiment Results

3.1 Experiment Setting

Our experiment environment consists of 24 physical servers, each with the following hardware specifications: quad-core X5460 CPU * 2 with hyper-threading, 16 GB memory, and 250 GB disk. The hypervisor is Xen 4.1 and the OS of domain 0 is

Gentoo. There are three kinds of VMs: the auto-scaling master (which manages the running VM cluster), the running VMs, and the data storages that runs MySQL servers. All the VMs use Gentoo OS. The configurations are as follow: Auto-scaling master: 2 core, 2G memory, and 4G disk space; Running VM: 4 core, 4G memory, and 4G disk space; Data storage: 1 core, 4G memory, 100G disk space.

MediaWiki [10] is used as the application benchmark in our experiments. MediaWiki is an open source wiki package originally for use on Wikipedia. We set up MediaWiki and create web pages to simulate a web application uploaded by a user. The contents are the dumps from Wikipedia. The web pages are set to read-only mode.

We use httperf [5] as our performance measuring tool. *Httperf* is a robust and well-known tool for measuring web server performance. It can generate various HTTP workloads, and measure the performance such as average response time.

The workload we used in the experiment is PREDICTABLE. PREDICTABLE is the workload from the log of *Judgegirl*, an online grading system for teaching purpose in department of CSIE, NTU. In the record, each data point represents the load in fifteen minutes. We shrink this length into thirty seconds and the result is shown in Figure 3. There is a pattern that appears four times in Figure 3, each of which is similar but is slightly different from the others. Also there are some sudden workload changes within a pattern.

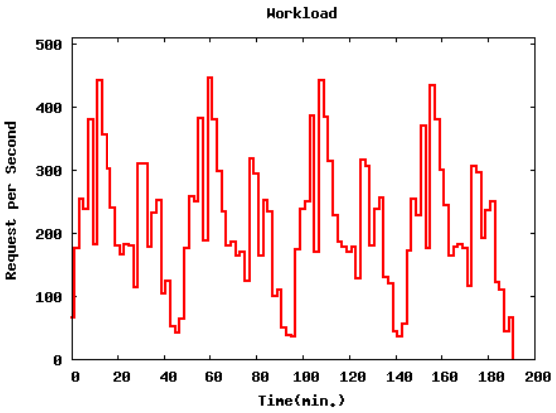

Fig. 3. PREDICTABLE workload

3.2 Comparison of Scaling Algorithms

For some web applications, such as stock market or enterprise applications, there exist periodic behaviors. In this experiment, we use PREDICTABLE workload, which has repeated behavior patterns, to test our auto-scaling algorithm. Figure 3 shows the workload. The load interval is two minutes. We compare three scaling strategies: majority vote with scaling threshold (30, 70), workload-based, and workload-based with trend analysis. The monitor intervals for all three algorithms are one minute. The number of database servers is fixed to one in the experiment.

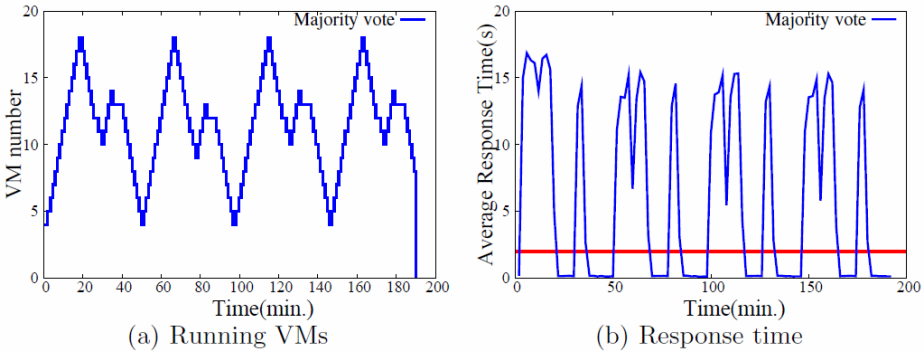

Fig. 4. Majority vote under PREDICTABLE workload

Figure 4(a) and Figure 4(b) are the results of majority vote. Even though the number of running VMs changes with workload, the average response time suffers a lot. The results show that majority vote is not an effective scaling algorithm for web applications with frequently changing workloads.

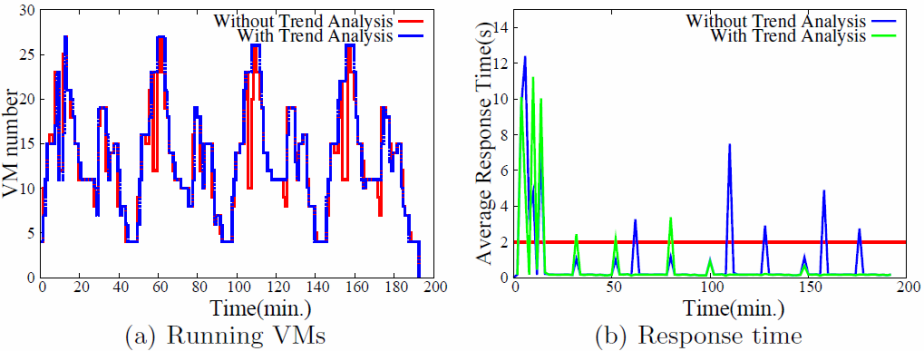

Fig. 5. Workload-based under PREDICTABLE workload

Figure 5(a) and Figure 5(b) depict the number of running VMs and average response time using workload-based as scaling algorithm. Workload-based outperforms majority vote. Furthermore, workload-based with trend analysis has better performance than without trend analysis. In Figure 5(b), there are five peaks (longer than 2 seconds) in the response time without trend analysis. These increasings are caused by sudden large workload changes. For example, in time 108, the workload drops, and the number of running VM decreases with it. However, in time 110, the load suddenly increases. Even if workload-based can respond to sudden workload increase, it still takes time to balance the load to these newly added VMs. Thus the average response time increases for a short period of time until the load is balanced.

On the other hand, workload-based with trend analysis takes the historical trend information into consideration while making scaling decisions. When there is workload fluctuating, it will not invoke scaling action. Therefore, it results in only two peaks in the response time (at time 50 and 80).

In summary, for workloads with periodic behavior, using workload-based algorithm with trend analysis performs the best among all three strategies. Slight sudden workload change will not affect workload-based algorithm with trend analysis. However, the cost of wrong analysis may be high.

3.3 Scaling Data Access Tier

In this section, we study the effect of auto-scaling on the data access tier. We add a backend VM with Round-Robin DNS. This VM also monitors the queries per second to the database servers, and make database scaling decisions using workload-based algorithm. The auto-scaling algorithm for running VMs is workload-based. Other settings are the same as previous section.

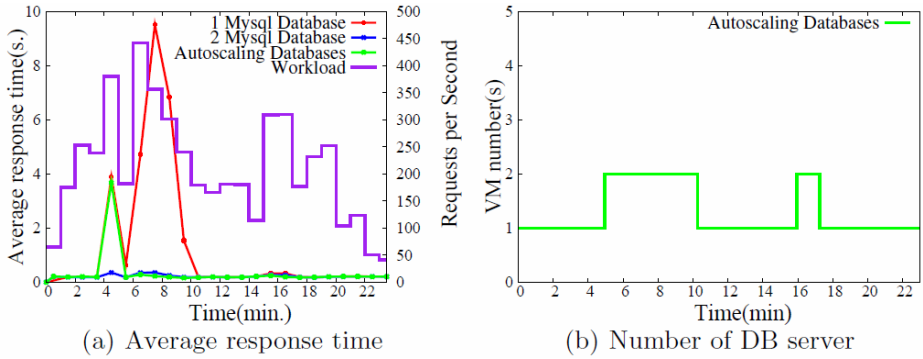

Fig. 6. Database tier

Figure 6(a) shows the performance result. The purple line is the workload. The red line shows the average response time of using only one database server. The response time drastically increases while the workload increases. On the other hand, the average response times of using two database servers always remain short no matter how workload changes.

As can be seen from Figure 6(a), auto-scaling with workload-based algorithm can always maintain low average response time. When the response time increases to almost 4 seconds at the fourth minute, the large amount of workloads trigger the auto-scaling system and a new database server is added to share the workload. The average response time decreases and remains short after then.

Figure 6(b) shows the number of database servers used. It is clear that the number of database servers is dynamically adjusted according to the workload. The experiment result shows that by applying auto-scaling to the data access tier, we can maintain low average response time while using the right amount of active database servers. By keeping fewer number of active database servers, energy consumption can be reduced.

4 Related Work

The auto-scaling feature has been provided in several cloud service providers and cloud computing systems, such as Amazon EC2 [13] and Google App Engine [2].

Auto scaling in Amazon EC2 is enabled by Amazon CloudWatch, which monitors the resource usage on user instances. Google App Engine [2] provides a very simple auto-scaling strategy. If the volume of incoming requests exceeds the capacity of the instances currently available, they will have to wait in the Pending Queue. When the number of pending requests exceeds a threshold value, a new instance will be created to share the workload.

Many softwares such as Scalr [3] and RightScale [4] provide auto-scaling mechanism that can apply to cloud environments like Eucalyptus [14] or Amazon EC2. Both Scalr [3] and RightScale [4] scales the number of VMs based on the workload on each back-end server. The scaling algorithms are presumed to be majority vote. In this paper, our empirical study has shown that majority vote is not effective for scaling workloads with periodic behaviors.

All of the above existing solutions do not address the issue of workload trend prediction. Most of them use majority vote to deal with workloads even if the workload is predictable. In contrast, our auto scaling system provides trend analysis algorithm and can scale out quickly.

The scaling decision algorithm plays a critical role in an auto scaling system. Chieu et al. [15] proposed an architecture for scaling based on predefined thresholds for Web applications. The algorithm scales out when all VM session numbers exceed the threshold. This approach is simple but insensitive to workload change. Our algorithm is more responsive to workload change since the decision is based on requests per second and HTTP response time and thus can accurately characterize Web application behavior. Mao et al. [16] presented a scaling approach to deal with batch jobs. According to the deadline of each job, it decides whether using current number of VMs is sufficient to meet the deadline. Since Mao's algorithm only considers batch jobs, it is not applicable to Web applications.

Another way to make scaling decision is by workload prediction. Caron et al. [11] used KMP algorithm to find patterns from history data based on N previous time interval. Gmach et al. [12] used an ARMA scheme to find periodgram function. The two prediction algorithms aim to predict the precise workload. However, their results show that it is difficult to make accurate prediction of precise workload. In contrast, our analysis algorithm only predicts the trend of workload change for two reasons. First, predicting workload trend requires much less time complexity than predicting precise workload. Second, our experiments demonstrate that workload-trend guided scaling is very effective.

5 Conclusion

Auto-scaling technique provides on-demand resources according to workload in cloud computing system. In this work, we propose an effective scaling algorithm, *Workload Based algorithm*, for 3-tier web applications. We compare the effectiveness of two scaling algorithms – *majority vote* and *workload-based*. *Majority vote*, a simple scaling strategy used in most existing systems, makes scaling decisions according to the load of each running VM, while *workload-based* uses the incoming workload, which is requests per second in our work, as the criteria for making scaling decisions.

A helper algorithm, *Trend prediction*, is devised to deal with workloads that exhibit periodical behaviors.

We conduct experiments to evaluate the performance of different scaling algorithms. We compared the performance of these algorithms under actual workload with periodical behavior. The results show that for workloads with periodical behavior, using workload-based algorithm with trend analysis performs the best among all three strategies. Slight sudden workload change will not affect workload-based algorithm with trend analysis. We also show that applying auto-scaling to data access tier can reduce the total database server used while maintaining the performance.

References

1. Amazon elastic compute cloud, http://aws.amazon.com/ec2/
2. Google app engine, https://developers.google.com/appengine/
3. Scalr, http://www.scalr.net/
4. Rightscale, http://www.rightscale.com/
5. Mosberger, D., Jin, T.: httperf - a tool for measuring web server performance. SIGMETRICS Perform. Eval. Rev. 26(3), 31–37 (1998)
6. Urdaneta, G., Pierre, G., van Steen, M.: Wikipedia workload analysis for decentralized hosting. Comput. Netw. 53(11), 1830–1845 (2009)
7. Arlitt, M., Krishnamurthy, D., Rolia, J.: Characterizing the scalability of a large web-based shopping system. ACM Trans. Internet Technol. 1(1), 44–69 (2001)
8. Davison, B.D.: Learning web request patterns (2004)
9. Wang, H., Li, B.: Shrinking tuning parameter selection with a diverging number of parameters. Journal of the Royal Statistical Society 71(3), 671–683 (2009)
10. Mediawiki, http://www.mediawiki.org/
11. Caron, E., Desprez, F., Muresan, A.: Forecasting for grid and cloud computing on-demand resources based on pattern matching. In: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CLOUDCOM 2010), pp. 456–463 (2010)
12. Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Workload analysis and demand prediction of enterprise data center applications. In: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization (IISWC 2007), pp. 171–180 (2007)
13. Amazon auto scaling, http://aws.amazon.com/autoscaling/
14. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2009), pp. 124–131 (2009)
15. Chieu, T., Mohindra, A., Karve, A., Segal, A.: Dynamic scaling of web applications in a virtualized cloud computing environment. In: Proceedings of the 2009 IEEE International Conference on e-Business Engineering (ICEBE 2009), pp. 281–286 (2009)
16. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID 2010), pp. 41–48 (2010)