A Queueing-based Model for Performance Management on Cloud

Hao-peng CHEN[1,2], Shao-chong LI[1]
[1] School of Software, Shanghai Jiao Tong University, Shanghai, China
[2] School of Computer Science, Georgia Institute of Technology, Atlanta, USA
E-mail: chen-hp@sjtu.edu.cn, lee.shaochong@gmail.com

Abstract—**Cloud computing is a new trend for computing resource provision. Many public clouds are available for developers to transfer or build web applications on cloud. As a result, the computing resource scheduling and performance managing have been ones of the most important aspects of clouding computing. In this paper, we propose a queueing-based model for performance management on cloud. In this model, the web applications are modeled as queues and the virtual machines are modeled as service centers. We apply the queueing theory onto how to dynamically create and remove virtual machines in order to implement scaling up and down. There is no VM (Virtual Machine) live migration involved in this model which makes it much simpler than some existing models. The result of case study has shown this model is effective to scaling up and down. The more precise measurement and analysis of this model will be done in future.**

Keywords- performance management; cloud; queueing

I. INTRODUCTION *(HEADING 1)*

Cloud computing has been an emerging technology for provisioning computing resource and providing infrastructure of web applications in recent years. Cloud computing greatly lowers the threshold for deploying and maintaining web applications since it provides infrastructure as a service (IaaS) and platform as a service (PaaS) for web applications [1]. Consequently, a number of web applications, particularly the web applications of medium and small enterprises, have been built into a cloud environment. Meanwhile, leading IT companies have established public commercial clouds as a new kind of investment. For example, Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers [2]. Google App Engine enables enterprises to build and host web applications on the same systems that power Google applications. App Engine offers fast development and deployment; simple administration, with no need to worry about hardware, patches or backups; and effortless scalability [3]. IBM also provides cloud options. Whether you choose to build private clouds, use the IBM cloud, or create a hybrid cloud that includes both, these secure workload solutions provide superior service management and new choices for deployment [4]. We even can establish a private cloud with Ubuntu Enterprise Cloud to offer immediacy and elasticity in the infrastructure of web applications [5]. In summary, both of the numbers of cloud applications and providers have kept gradually increasing for a couple of years. As a result, computing resource scheduling and performance managing have been ones of the most important aspects of clouding computing.

Among the top 10 obstacles of cloud which the report [1] proposes, the obstacle 8, Scaling Quickly, is our focus. When a number of web applications are deployed into a cloud environment, dynamical allocating the computing resource to web applications on demand has a positive effect not only on the performance of web applications, but also on the energy saving. The solution to eliminate this obstacle is to automatically scale quickly up and down in response to load in order to save money for web applications providers by optimizing the requesting of computing resource, but without violating service level agreements [1]. Meanwhile, the cloud providers also can save money by optimizing the allocation of computing resource and saving energy, since the cloud providers needn't to provide excessive active computing resources. To achieve the aim of dynamic scaling, we need proper tools and models to diagnose the runtime requirements of web applications.

Since there is not any standard model has been widely accepted by industry yet, scaling up and down is an open issue for researchers. The cloud providers, such as Amazon, IBM, and Google have their own mechanisms which are commercial ones and inherited from their existing proprietary technology. The researchers from universities and institutes also have proposed some models and methods. For example, in [6], the author introduces many outcomes on predicting system performance based on machine learning obtained in RAD lab of University of California at Berkeley. The existing solutions to scaling up and down are designed via various techniques, such as statistical methods, machine learning, and queueing theory.

Aware of the advantages and disadvantages of these solutions, we propose a queueing-based model for performance management on cloud. In this model, the web applications are modeled as queues and the virtual machines are modeled as service centers. We apply the queueing theory onto how to dynamically create and remove virtual machines in order to implement scaling up and down. The remainder of the paper is structured as follows. Section II briefly summaries the related works; Section III gives the principle of our model; Section IV describes the mechanism for scaling up and down; Section V

This paper is supported by the Project of Daystar of Shanghai Jiao Tong University.

shows the case study of our model; and conclusion in Section VI.

II. Related Work

Performance management on a cluster, an issue much similar to performance management on cloud, has been adequately discussed in past. For example, in [7], the authors aimed to online response time optimization of Apache Web Server and analyzed the advantages and disadvantages of Newton's Method, fuzzy control, and heuristic method for optimization. But the performance management on cloud differs from the one on a cluster mainly in the following two aspects:

1. Since the web applications run into the virtual machines on cloud, such as Xen [8] and KVM [9], they are allowed to be heterogeneous. On the contrary, the web applications deployed on a cluster must be homogeneous due to the unified software platform on each node.
2. It is via virtual machine that the web applications share computing resource on cloud. It is much more complex than the mechanism of resource sharing on a cluster which has no intermediary.

Aware of the differences between the performance management on cloud and on a cluster, many researchers study on this issue. For example, RAD Lab of Berkeley focuses on the pervasive and aggressive use of statistical machine learning as a diagnostic and predictive tool that would allow dynamic scaling, automatic reaction to performance and correctness problems, and generally automatic management of many aspects of these systems. In [10], Kernel Canonical Correlation Analysis (KCCA) is used to predict the execution time of MapReduce jobs in a data-intensive system running on a cloud. The reason why KCCA is used has been given in their earlier work published in [11].

In [12], the authors proposed a Queueing Theory based method to predict the performance of the service exposed by the cloud. Although the correctness of the method has been demonstrated by some experiments and simulations, the model they set up is quite simple due to its presumption that a cloud only exposes one service. Actually, the authors only propose a generalized method to analyze and predict the performance of a service. They did not consider the special context of Cloud Computing. In our opinion, the structure of a cloud is like a multiple Queues but not a single Queue.

It is inevitable that there is random error between predicted status and real-time one though prediction is a feasible and effective way to optimize the usage of computing resource. So prediction is a risky method which is possible to result in a serious situation. Moreover, the predicting needs enough samples to be learned which is not available for new deployed applications. Thus, the initial resource provisioning is hardly done by prediction. So some researchers focus on the real-time status based performance management. In [13], the authors propose a packing algorithm based method to minimize the number of running machines, so as to save energy. It can be a reference model for dynamical scheduling. Once a VM is shutdown, a VM is started, or a VM is resized, the method will rearrange the locations of all the VMs by live migration to minimize the number of running machines and shutdown the unnecessary machines. But it has two obvious flaws. The first one is excessive cost of live migration. Since the packing algorithm is used, in the extreme situation, almost all the VMs need to be migrated. On the one hand, live migration itself is a time-cost task so that people shutdown the VM and start a new VM instead in practice. On the other hand, the number of the VMs needs to be migrated can be over control. The second flaw is that the over-provision approach they use is a waste. In order to avoid the frequent VM resizing events, the authors set a configurable parameter α . When a VM requires the computing power of P, they allocate $(1+\alpha)P$ power to the VM. Thus, when the VM resizing is in the range from $(1-\alpha)P$ to $(1+\alpha)P$, it does not trigger a resizing event. The computing power is wasted by the factor α . Moreover, if the P requested by a VM is greater than all power of a single machine, it is difficult to allocate the power to the VM. However, this model could be a reference to dynamical scheduling.

VMs joining and leaving is involved in scaling up and down. The key is to make all the VMs of an application know the changes of membership. In [14], a membership scheme is designed to manage membership discovery & management on EC2. This scheme use Amazon announced two new features—Availability Zones & Elastic IP Addresses—to achieve their aims. Either membership of application tier or membership of database tier is can be discovered and managed dynamically, and the load balancer can rescheduling tasks according to the current membership. This scheme ensures that the member joining and leaving in scaling up and down has not negative impact on task scheduling.

There are still many other researches on the performance management on cloud. All the researches can be roughly divided into two kinds: by prediction and by real-time status. As our analysis, they all have pros and cons. We design a model based on the real-time status of web applications which more exactly captures the structure of cloud and avoids frequently live migration of VM.

III. Principle of the Model

A. The queue model of web applications

Performance, to put it simply, is how quickly the web application can respond to a given logical operation from a given individual user. Response time is a measure of the amount of time the application consumes while processing a client request [15]. Since the response time is an important measure of application performance, we can use it to determine whether the specified web application violates the service level agreements. In a service level agreement, such as a WSLA agreement [16], the accepted range of response time will be described as an item.

To capture the structure of web applications on cloud, we figure out that the process of client requests in a single web application has the following features:

1. The interarrival times between any two successive client requests are independent of each other and have a common probability distribution.
2. The clients will receive responses if requests are processed by web application in time, or receive exception(s) due to the timeout of waiting. They even can abort the requests as their wills.
3. Because a web application usually exposes multiple services to the users, the service time needed for a request is dependent on the real-time status of the application and the service it invokes. All the service times have identically probability distribution. Furthermore, they are independent of interarrival times.
4. The client requests can be served in many possible orders, such as first come first served, last come first served, shortest processing time first, random order, round robin, and so on. However, the first come first served is still the predominant order.
5. The cloud platform can create a single VM or a cluster of VMs to process the client requests. Thus, there are several possible kinds of application capacity.
6. Since the cache(s) or buffer(s) of the VM(s) of an application is finite, the number of waiting requests is limited. It means if the waiting room of an application is fully occupied, when extra requests arrive at this service, they would be lost.

The above six features have shown that the model of client requests processing is a typical queueing model, so we can resort to queueing theory to capture the structure of web applications.

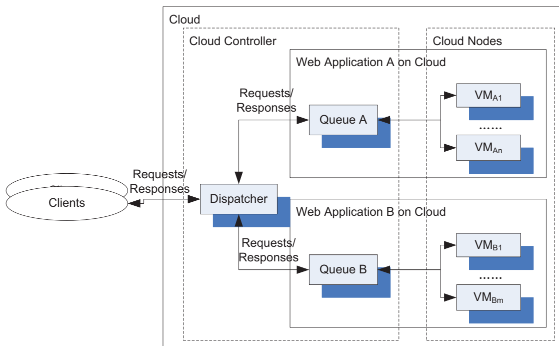

Figure 1. Queue model of web application on cloud

When a web application is deployed on a cloud, the Cloud Controller, as the portal of cloud, will establish a queue for it to hold the client requests. Meanwhile, a certain number of VMs will be created by Cloud Controller on Cloud Node(s). The number of initially created VMs can be specified by service level agreement or by empirical value in case of no constraint on it in service level agreement. The number of live VMs at runtime will vary with the dynamical creation and remove of VMs. All the VMs of a web application can run on either a single Cloud Node or multiple Cloud Nodes, and each of them has been allocated with the same computing resource. Regardless the physical location of VMs, there is an instance of the web application running into each of them. As a result, all the VMs compose a cluster and process the requests in the corresponding queue.

When a client sends a request to a web application on cloud, the request will be sent to the Cloud Controller. The dispatcher in Cloud Controller forwards the request to the queue of the target web application. The instances of the target web application running into VMs act as service centers to process the requests in the queue.

For example, in Figure 1, web application A has n instances each of which is running into a VM. Similar to A, web application B has m instances. All the VMs of A and B can run on a single Cloud Node or run on multiple Cloud Nodes, and they compose two clusters. Queue A and Queue B on Cloud Controller respectively associate with web application A and B.

The queue type is determined by its performance measurements, such as the probability distribution of the waiting time and the sojourn time of a request, the probability distribution of the amount of work in the application, and the probability distribution of the busy period of the application [17]. As most web applications, in a web application on cloud, the number of client requests and the service time are random variables, so we can consider they has Poisson or exponential distribution. By convention, we use M to respectively indicate the number of requests and the service time. Since each web application on cloud has one or multiple instances running into VM(s), and each instance can serve a certain number of requests. Thus, the number of the requests concurrently processed by a web application can be determined. By convention, we use S to indicate this number. For any application, its capacity is limited. So it has the upper limitation of the number of clients they can serve. This upper limitation is the sum of S and the number of waiting requests the queue can hold. By convention, we use k to indicate the capacity. Thus, the queue model for the web application on cloud is abstracted as an *M/M/S/k* one.

B. Computing resource management on cloud

Since the web applications are modeled as queues and the VMs are modeled as service centers, we can dynamically create and remove VMs according to the number of necessary service centers in order to scale up and down. To achieve this aim, we give the design as Figure 2.

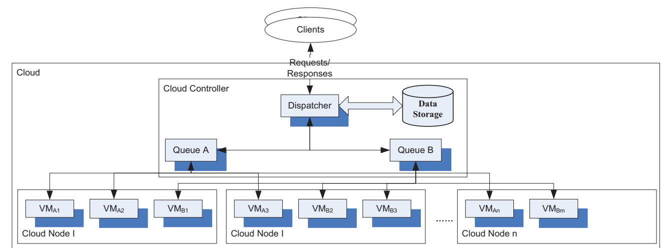

Figure 2. Queue model of web application on cloud

Firstly, we allocate same amount of computing resource to all VMs. Since all the Cloud Nodes usually have the same hardware configuration, we can specify a fix hardware allocation policy to all VMs. For example, we specify that each Cloud Node will run at most three VMs. Thus, each VM will get one third computing resource of a Cloud Node, including CPU, memory, hard disk, and net bandwidth. This policy for resource allocation can effectively avoid wasting computing resource. Meanwhile, it guarantees that each service center of a web application will be identical. When a new service center need to be create, we just need to find a Cloud Node which has spare resource and create it without worry about resource overprovision or underprovision. For example, in Figure 2, if we have to create a new VM for application A, we can create it on Cloud Node n since it has spare resource. After we create the new VM, the resource on Cloud Node n will be totally allocated to VMs without any wasting. When a VM is removed, the resource it releases is exactly enough for creating a new VM. With this policy, when the performance of a web application is not acceptable, the new VM(s) will be created to improve it and no live migration of VM is necessary.

Secondly, the response time of each request and the length of each waiting queue will be recorded into data storage by dispatcher of Cloud Controller. Note that the sojourn time of a request is the waiting time plus the service time, which equals to response time. Because the Cloud Controller is the portal of cloud, all the requests will be sent to dispatcher and forward to queues, and all the responses will be sent back to clients via dispatcher. As a result, the dispatcher can record the response time exactly. When requests arrive at cloud, the dispatcher intercepts all of them, records their arrival times into the data storage, and updates the number of new arriving requests during the unit time in the data storage. Subsequently, the dispatcher dispatches the requests to the queues. When a response is generated and sent back to client, the dispatcher intercepts it again to read the departure time. Subsequently, the dispatcher updates the number of departing requests during the unit time in the data storage. Lastly, the response is delivered to client.

Thirdly, the Cloud Controller periodically checks whether the dynamical creation and remove of VM(s) is necessary. If a web application reaches its steady state, its expected number of requests waiting for serving, expected waiting time of requests, and expected sojourn time of requests will be steady. Furthermore, the distribution of these variables is independent of time; it means in any period or at any time, the distribution of these variables is the same, and the values of these variables before time t would not have impact on the values of these variables at time t. This property is called memoryless property. Thus, when each period expires, the Cloud Controller will accesses the data storage to retrieve the records and calculates the necessary number of VMs according to the performance of web application. Then the Cloud Controller will create or remove VMs and empty the data storage. Since the data stored in data storage is not very much, and it needs to be accessed frequently, it should be designed as an in-memory object, such as an in-memory database. The algorithm for determining how to dynamically create and remove VMs will be described in Section IV.

IV. Mechanism for Scaling Up and Down

A. *Dynamical creation and remove of VMs*

The template is designed so that author affiliations are not repeated each time for multiple authors of the same affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization). This template was designed for two affiliations.

We require the web application providers describe expected number of requests waiting for serving, expected waiting time of requests, and expected sojourn time of requests in Service Level Agreement as the constraints on performance. Meanwhile, they also need to specify an acceptable nonnegative error e and an acceptable number of successive failed periods n.

The Cloud Controller periodical checks the data storage and determines whether the dynamical creation and remove of VMs is necessary. Usually, there are multiple web applications deployed on cloud. The Cloud Controller will iteratively process all the web applications. For each application, the Cloud Controller calculates the mean number of requests waiting for serving, the mean waiting time of requests, and the mean sojourn time of requests in a period which contains certain amount of unit time, and compare these real-time variables with the expected variables specified in Service Level Agreement. There must be error between these two set of variables, especially we focus on the error between the expected sojourn time and the real-time mean sojourn time. If the error exceeds e for n times, we consider the web application deviates its steady state, and as a result, it cannot satisfy the performance requirement any longer. Then the necessary number of VMs is calculated and the VMs are dynamically created. If the error doesn't exceed e for $2n$ times, we consider the web application reaches its steady state, and maybe it has been overprovisioned. Then the necessary number of VMs is calculated and the VMs are probably dynamically removed.

The periodical process is as the following pseudo-code:

```
1    function Scaling_Up_Down (Collection apps)
2     // apps: all the web application deployed on cloud
3     Static float e, expectedSojournTime;
4     // e: an acceptable nonnegative error between expected
5     // sojourn time and real-time mean sojourn time.
6     // expectedSojournTime: the expected sojourn time
7     // specified in Service Level Agreement.
8     Static int n;
9     // n:an acceptable number of successive failed periods.
10    Static int timesG, timesB = 0;
11    // timesG: the number of successive successful periods
12    // timesB: the number of successive failed periods
13    for each application in apps
14       Collection his = getHistory(application);
15       // his: history records of application in data storage
16       int meanSojournTime = CalculateSojourn(his, S);
17       // meanSojournTime: mean sojourn time in a period
18       // S: the number of instance of application
19       if (meanSojournTime - expectedSojournTime) > e
20          timesB++;
21          timesG = 0;
```

```
22          if timesB > n
23            int number = CalculateVMs(application, 1);
24            // Calculate the necessary number of VMs
25            CreateVMs (number - existing);
26            //existing: the number of existing VMs of application
27          endif
28        else
29          timesB = 0;
30          timesG ++;
31          if timesB > 2n
32            int number = CalculateVMs(application, -1);
33            // Calculate the necessary number of VMs
34            RemoveVMs (existing - number);
35            //existing: the number of existing VMs of application
36          endif
37        endif
38    endfor
39  endfunction
```

The functions CreateVMs() and RemoveVMs() are system functions provided by cloud platform to create and remove VMs. The function getHistory() is provided by the data storage to retrieve data. The function CalculateSojourn() and CalculateVMs() will described in Section IV.B.

B. Queueing algorithm for calculating necessary VMs

The CalculateSojourn() is periodically invoked by dispatcher to calculate the real-time sojourn time. We set the unit time as 1 second, so in every second, the dispatcher records the number of requests arriving in the web application, indicated as λ_i; the number of requests departing from each instance of the application, indicated as $\mu_{i,j}$. For every period, such as every 100 seconds, we calculate the real-time mean sojourn time as the following pseudo-code:

```
1   function CalculateSojourn (Collection his, S)
2     // his: history records of application in data storage
3     // S: the number of instance of application
4     get all λ_i and μ_i,j from his;
5     // the value range of j is from 0 to S
6     float λ = average( λ_i )
7     // calculate the average number of requests arrived
8     float μ = weightedAverage( μ_i,j );
9     // calculate the weighted average number of requests
10    // departing from each instance of application.
11    // P_0 : the probability of the queue with length of 0
12    for j =1 to k
13       P_j = Queueing( λ , μ , P_0 ,j);
14       // P_j : the probability of the queue with length of j
15    endfor
16    int L = Sum(j* P_j , k);
17    // L: the real-time mean length of queue
      // k: the value range of j is from 0 to k
18    λ_e = λ(1 − P_k)
19    // λ_e is effective request arrival rate
20    W = (1/λ_e) L
21    //W: the real-time sojourn time
22    return W;
23  endfunction
```

The function Queueing() is a standard function provided by queueing toolkit to calculate the probabilities of the queue with length from 0 to n.

The function CalculateVMs() is used to calculate the necessary VMs. Its process is as following pseudo-code:

```
1   function CalculateVMs (App application, Flag f)
2     // application: the target for calculating
3     // f: a flag. 1 indicates need more VMs, -1 indicates need
4     // less VMs.
5     int existing = getExistingVMs(application);
6     //existing: the number of existing VMs of application
7     for (i = existing; i > 0 and i <= limit; i=i+f)
8     // limit: the upper limitation of the number of application
9         // which can be specified in Service Level Agreement
10        float W = CalculateSojourn(his, i)
11        // calculate the sojourn time when application has i
12        // instances
13        if abs(meanSojournTime - expectedSojournTime) < e
14           break;
15        endif
16     endfor
17     return number;
18  endfunction
```

The result returned by CalculateVMs() is the minimum of the number of VMs which can guarantee the performance of web application described in Service Level Agreement.

V. CASE STUDY

Since we are studying on service registry which is deployed as a cloud application, we shared the experimental environment with it. The shared experimental environment is a 7-computer environment in which we runs one Cloud Controller, two Cluster Controllers, and four Cluster Nodes. All the computers have same hardware configuration. The main features are:

- Product: Dell Inc. Inspiron 531
- CPU: AMD Athlon(tm) 64 X2 Dual Core Processor 5200+
- Memory: DIMM 667 MHz 1GiB * 2
- Hard Drive: Seagate ATA Disk 232GiB (250GB)

We set up a policy that there are at most 3 VMs running on each computer and each VM will obtain one third computing resource of a computer.

We deployed two simple web applications on the experimental environment. The first one is an online bookstore, which provides services for logging in, browsing book, and selling book. This book store has 1,000 kinds of books and 500 registered users. The second one is an online box office, which provides services for logging, browsing tickets information,

and booking tickets. This box office has 500 kinds of tickets and 1,000 registered users.

For simplification, we initialized these two applications as M/M/3/3 model. So we create 3 VMs on Cloud Node I for online bookstore and 3 VMs on Cloud Node II for online box office.

We simulate 200 users to randomly send requests to the two web applications to invoke the services they provide. We record the number of requests arriving in the web applications λ_i and the number of requests departing from the applications $\mu_{i,j}$ every second for 2,000 seconds. We find the two applications have close performances.

We set the same parameters for both of the applications, including the lower performance of them as the acceptable performance, a tiny value, 50 ms, as acceptable error, 100 seconds as the length of period, and 5 as the acceptable successive failed periods. Then, we simulate 150 users to send requests to the online Bookstore and 50 users to send requests to the online Box Office. After 6-8 periods in experiments, a new VM is created on a new Cloud Node, indicated as Cloud Node III, for running a new instance of online Bookstore.

Then, we reset the experiment to the beginning configuration, that is, the 200 users randomly send requests to the two web applications. We can find after 6-9 periods in experiments, a VM of online Bookstore will be removed. The removed VM may be on Cloud Node I or Cloud Node III when we repeat the experiment.

The result of experiment has shown that our queueing based model is effective to scale up and down the web applications on cloud and no VM live migration is involved. However, our experiments are not enough to precisely measure the effect of our model on usage of computing resource, because the hosting environment we establish is an experimental one which is quite different from the true cloud platform and there is no real user to access it. As a result, we have no enough valid data to do more analysis. The concurrent users we simulate are not massive enough to check the efficiency of our model. After all, when the amount of concurrent users is not massive enough, the improvement of performance is not notable. So in next work, we will deploy it into a commercial environment and precisely measure and analyze it.

VI. CONCLUSION

In this paper, we propose a queueing-based model for performance management on cloud. In this model, the web applications are modeled as queues and the virtual machines are modeled as service centers. We apply the queueing theory onto how to dynamically create and remove virtual machines in order to implement scaling up and down. There is no VM (Virtual Machine) live migration involved in this model which makes it much simpler than some existing models.

The result of experiment has shown that our queueing based model is effective to scale up and down the web applications on cloud and no VM live migration is involved. However, our experiments are not enough to precisely measure the effect of our model on usage of computing resource. So in next work, we will deploy it into a commercial environment and do more precise measurement and analysis.

ACKNOWLEDGMENT

As a visiting scholar of Georgia Institute of Technology, Hao-peng Chen thanks Georgia Institute of Technology and Prof. Ling Liu, a Professor of College of Computing at Georgia Institute of Technology, for their supports.

REFERENCES

[1] Michael Armbrust, Armando Fox, et.al, Above the Clouds: A Berkeley View of Cloud Computing, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, 2009

[2] Amazon, *Amazon Elastic Compute Cloud (Amazon EC2)*, available at: http://aws.amazon.com/ec2/, 2010

[3] Google, *Google App Engine*, available at : http://code.google.com/intl/en/appengine/, 2010

[4] IBM, IBM Smart Business Cloud Computing, available at: http://www.ibm.com/ibm/cloud/, 2010

[5] Ubuntu*, Private cloud: Ubuntu Enterprise Cloud*, available at: http://www.ubuntu.com/cloud/private, 2010

[6] Archana Sulochana Ganapath, *Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning*, Technical Report No. UCB/EECS-2009-181, available at: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-181.html, December 17, 2009

[7] Xue Liu, Lui Sha, Yixin Diao, Steven Froehlich, Joseph L. Hellerstein, and Sujay Parekh, *Online Response Time Optimization of Apache Web Server*, IWQoS 2003, LNCS 2707, pp. 461–478, 2003.

[8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. *Xen and the Art of Virtualization*, in Proc SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.

[9] *Kernel Based Virtual Machine*, abailable at: http://www.linux-kvm.org/page/Main_Page, 2010

[10] Ganapathi, A., Y. Chen, A. Fox, R. Katz, and D. Patterson, *Statistics-Driven Workload Modeling for the Cloud*, in Proc Workshop on Self-Managing Database Systems (SMDB), 2010

[11] A. Ganapathi, H. Kuno, U. Daval, J. Wiener, A. Fox, M. Jordan, and D. Patterson, *Predicting Multiple Performance Metrics for Queries: Better Decisions Enabled by Machine Learning*, in Proc International Conference on Data Engineering, 2009.

[12] K. Xiong and H. Perros, *Service performance and analysis in cloud computing*, ICWS 2009, in Proc International Workshop on Cloud Computing, July, 6-10 (2009), LA.

[13] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, Liang Zhong, *EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments*, in Proc 2009 IEEE International Conference on Cloud Computing, 2009

[14] Afkham Azeez , *Autoscaling Web Services on Amazon EC2*, http://people.apache.org/~azeez/autoscaling-web-services-azeez.pdf

[15] Ted Neward, *Effective Enterprise Java*, Addison Wesley Professional, Boston, August 26, 2004

[16] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, Richard Franck, *Web Service Level Agreement (WSLA) Language Specification*, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, 2003/01/28

[17] Ivo Adan, Jacques Resing, *Queueing Theory*, February 28, 2002, available at: www.win.tue.nl/~iadan/queueing.pdf