Estimating Service Response Time for Elastic Cloud Applications

Khaled Salah
Electrical and Computer Engineering Department
Khalifa University of Science, Technology and Research (KUSTAR), UAE
Email: khaled.salah@kustar.ac.ae

Raouf Boutaba
David R. Cheriton School of Computer Science,
University of Waterloo, Canada, and the Division of IT Convergence Engineering, POSTECH, Pohang, Korea
Email: rboutaba@cs.uwaterloo.ca

***Abstract*--** **This paper presents a Markovian analytical model to estimate service response time for elastic cloud applications. Given the expected application workload, the number of virtual machine (VM) instances, and the capacity of each VM instance, the model can approximate the mean service time. The mean service time is a critical metric to estimate, and contributes to the SLA end-to-end response time experienced by application users. The end-to-end response time is an aggregated delay of the service time in addition to delays incurred at the network nodes and links. Our analytical model focuses on estimating the mean service time; however, the model is sufficiently general and can be extremely useful in studying cloud performance. Equations for key performance measures are derived. These measures include mean response time, throughput, request loss, queueing probability, and CPU utilization. The correctness of the model has been verified using discrete-event simulation.**

KEYWORDS: **Cloud Computing, Elastic Applications, Service and Network Delays, SLA, Queueing Theory, Performance Modeling and Analysis**

I. INTRODUCTION

In cloud computing, the response time is a key QoS performance criterion for elastic cloud applications. The response time is one of the major parameters that get specified in the SLA (Service Level Agreement), and it must be satisfied by the cloud provider [1, 2]. Elastic cloud applications are scalable applications hosted by the cloud. The elasticity of cloud applications is accomplished by continuously monitoring the application workload and provisioning (or auto-scaling) accordingly the needed cloud nodes/resources (or VM instances). Examples of elastic applications include web services, financial services, multimedia systems, and HPC (High Performance Computing) applications such as bioinformatics, astronomy, medical imaging, oil and gas exploration, etc.

At the cloud datacenter, as depicted in Figure 1, elasticity is typically performed by the Cloud Controller (CLC) or Fabric Controller (FC) in coordination with the Load Balancer (LB). The CLC or FC is responsible for managing, provisioning and de-provisioning the underlying cloud resources. Cloud resources are typically compute nodes which may require access to storage or database servers. Each compute node hosts a single or multiple computer instances or VM instances. The LB is responsible for communicating continuously with the provisioned VM instances to determine the load at each VM instance. The LB attempts to dispatch requests equally among all VM instances. For example, for web applications, the LB keeps track of the available web worker threads on each VM instance, and it dispatches the requests equally among those available threads [3,4]. In addition, the LB monitors the health and the state of each VM instance and informs the FC to provision (scale up) or de-provision (scale down) VM instances in order to meet the SLA criteria.

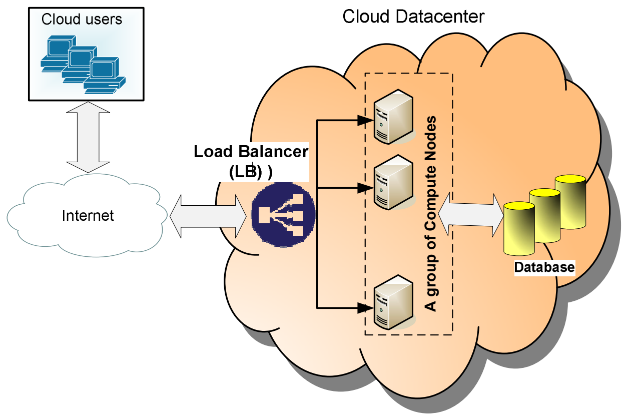


Figure 1. A typical elastic application hosted on a cloud datacenter

The cloud provider has to predict accurately the *minimum* required cloud resources to provision, especially those of compute or VM instances. This is necessary so that SLA performance criteria can be satisfied and resources can be utilized efficiently. Trial and error or over provisioning of compute instances are not desirable, as they may lead to SLA violation or result in a poor utilization of cloud resources and high system cost. More importantly, under heavy load and in the situation of encountering a violation of SLA requirement, an *individual* or *group* VM instances get provisioned and instantiated to satisfy the violated SLA criterion (as in Amazon Auto Scale [3,4]). If the size of this group is not accurately predicted, this may lead to huge incurred response times due to the time it takes to provision and instantiate new VM instances. Such instantiation time is estimated to be in the order of minutes [5]. Therefore, accurate prediction of the size and the number of compute instances for elastic applications becomes a critical performance and resource management issue.

In this paper, we present an analytical model that approximately determines the *minimum* required cloud compute instances to satisfy the specified SLA response time. Specifically, for a known or measured capacity (or service/processing rate) of VM instances and a monitored incoming load (or arrival request rate), the model can predict the overall service response time of an elastic application hosted on a given number of VM instances. We refer to the service response time as the sojourn time which includes processing/execution time and queueing or waiting time at the datacenter. This sojourn time is a part of the end-to-end response time which can include other link and node network delays [6,7].

Prior related work reported in [6-12] employs general queueing models (as those of *M/M/1*, *M/G/1*, *M/M/m*, *M/G/m/K*, or Erlang formulas) to capture and analyze the behavior of cloud systems and applications. None of these models is focused on determining the number of VM instances required to meet the service response time for elastic applications. Also, these models fall short of capturing the real behavior of the elastic applications. Specifically, all of these models ignore the role of the LB. As stated earlier, the LB plays an important role in dispatching, monitoring, and tracking the availability of compute instances at the cloud datacenter. This processing time at the LB can be significant. Hence, any analytical model should account for the role of LB in order to accurately model behavior and performance.

Our proposed analytical model can be used to model other similarly-behaving systems as those of elastic applications. For example, the model can be used in auto-scaling algorithm (such that of Amazon AWS) in which the capacity and number of compute instances to be provisioned are determined based on the measured response time and current load. Currently in Amazon AWS, auto scaling is based on setting a lower and upper thresholds for the overall CPU utilization. Also, our model can be used to predict the number of Hadoop cluster nodes (of a certain size and capacity) required to schedule and execute a MapReduce job.

The rest of the paper is organized as follows. Section II describes our analytical model to capture the behavior of an elastic application hosted on a cloud. Section III discusses the correctness of our proposed analytical model. Section IV presents numerical examples for two key performance measures of an elastic application in terms of offered load. Finally, Section V concludes the study and outlines future work.

II. ANALYTICAL MODEL

The handling of an incoming request for an elastic application hosted on a cloud is illustrated in Figure 2. As shown, an arriving request gets first queued in a finite buffer and then dequeued by the LB with a mean service time $1/r$. The LB will dequeue the request for processing if and only if one of the compute instances is readily available to handle a new request. This could happen if the compute instance has been newly launched or it has just finished servicing a request. The compute instance has to notify the LB upon the completion of a request. We assume each compute instance can service and process dispatched requests from the LB with a mean service time $1/\mu$. We assume that the LB will distribute the load evenly among all *m* provisioned compute instances. In this way, if we assume the incoming request rate is λ, the portion of incoming rate to each individual compute instance will be λ/m. The departure rate of all instances is denoted by γ, which also represents the overall throughput of the system.

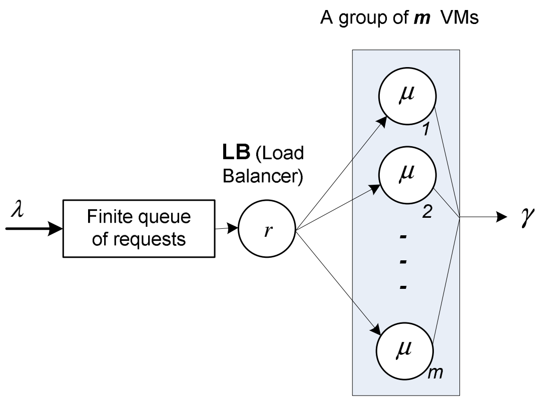


Figure 2. Finite queueing system model with a LB and *m* compute instances

In order to approximately model the behavior and performance of the above system, we assume that incoming requests follow a Poisson arrival λ, and all of the service times are independent and exponentially distributed with means of $1/r$ and $1/\mu$. Requests are serviced according to FCFS (First Come First Served) discipline. We also assume that $r \geq m\mu$. This is a reasonable assumption; otherwise, the LB will be a performance bottleneck. If $r < m\mu$, then the LB has to be re-sized and scaled up vertically by adding more CPU processing power, or scaled out horizontally by adding more LB instances.

Our analytical model uses the embedded Markov chain to represent the behavior of the queueing system, shown in Figure 2, with a state space $S = \{ (k,n), 0 \leq k \leq K, n \in \{0,1\} \}$, where k denotes the number of requests in the system and *n* denotes the type of processing taking place by either the LB or one of the compute instances. The queueing system has a buffer size of *K-m*. State (0,0) represents the special case when the system is empty. States $(k,1)$ represent the states where the request is being handled by the LB. States (*k*,0) represent the states where the request is being handled by one of the compute instances. The rate transition diagram is shown in Figure 3.

Let $q_{k,n}$ be the steady-state probabilities at state (*k,n*). A system of difference equations can be written as follows. At states (0,0), (1,0), and (1,1), we have

$$-\lambda q_{0,0} + \mu q_{1,0} = 0,$$

$$-(\lambda + \mu) q_{1,0} + r q_{1,1} = 0,$$

and

$$-(\lambda + r) q_{1,1} + \lambda q_{0,0} + 2\mu q_{2,0} = 0,$$

respectively.

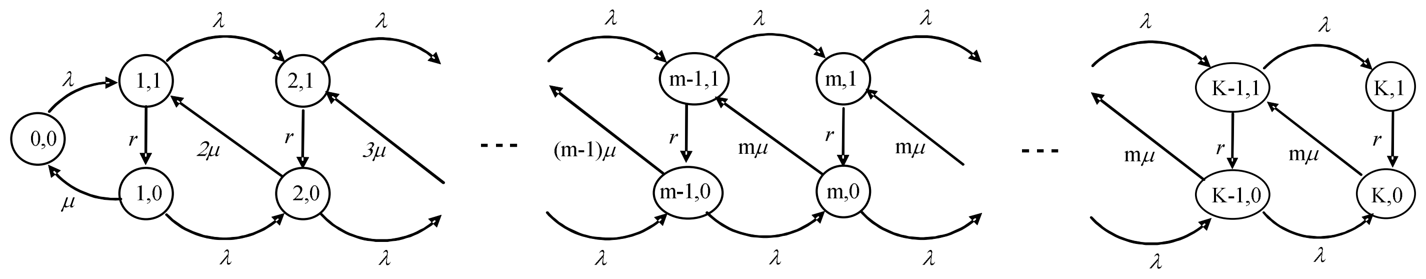


Figure 3. State transition diagram to capture handling requests for elastic applications

Therefore, the probabilities of $q_{1,0}$, $q_{1,1}$, and $q_{2,0}$ can be expressed as follows in terms of $q_{0,0}$, i.e.,

$$
\begin{aligned}
q_{1,0} &= \frac{\lambda}{\mu} q_{0,0}, \\
q_{1,1} &= \left(\frac{\lambda+\mu}{r}\right)\left(\frac{\lambda}{\mu}\right) q_{0,0}, \\
q_{2,0} &= \left(\frac{\lambda+r}{2\mu}\right) q_{1,1} - \left(\frac{\lambda}{2\mu}\right) q_{0,0}.
\end{aligned} \tag{1}
$$

At each state (k,1), the difference equation is expressed as

$$
\begin{aligned}
&-(\lambda+r)q_{k,1} + \lambda q_{k-1,1} + (k+1)\mu q_{k+1,0} = 0, \\
&\qquad 2 \le k \le m-1 \\
&-(\lambda+r)q_{k,1} + \lambda q_{k-1,1} + m\mu q_{k+1,0} = 0, \\
&\qquad k \ge m.
\end{aligned} \tag{2}
$$

At each state (k,0), the difference equation is expressed as

$$
\begin{aligned}
&-(\lambda+k\mu)q_{k,0} + r q_{k,1} + \lambda q_{k-1,0} = 0, \\
&\qquad 2 \le k \le m-1 \\
&-(\lambda+m\mu)q_{k,0} + r q_{k,1} + \lambda q_{k-1,0} = 0, \\
&\qquad k \ge m.
\end{aligned} \tag{3}
$$

Equation (3) can be rewritten as

$$
q_{k,1} = \begin{cases} \left(\dfrac{\lambda+k\mu}{r}\right) q_{k,0} - \left(\dfrac{\lambda}{r}\right) q_{k-1,0} & 2 \le k \le m-1 \\ \left(\dfrac{\lambda+m\mu}{r}\right) q_{k,0} - \left(\dfrac{\lambda}{r}\right) q_{k-1,0} & k \ge m \end{cases} \tag{4}
$$

Equation (4) can be rewritten as

$$
q_{k,0} = \begin{cases} \left(\dfrac{\lambda+r}{k\mu}\right) q_{k-1,1} - \left(\dfrac{\lambda}{k\mu}\right) q_{k-2,1} & 3 \le k \le m-1 \\ \left(\dfrac{\lambda+r}{m\mu}\right) q_{k-1,1} - \left(\dfrac{\lambda}{m\mu}\right) q_{k-2,1} & k \ge m \end{cases} \tag{5}
$$

The boundary probabilities at states (K, 1) and (K, 0) are as follows

$$-rq_{K,1} + \lambda q_{K-1,1} = 0,$$

and

$$-m\mu q_{K,0} + \lambda q_{K-1,0} + r q_{K,1} = 0,$$

respectively.

Therefore,

$$
\begin{aligned}
q_{K,1} &= \begin{cases} \dfrac{\lambda}{r} q_{K-1,1} & K > 1 \\ \dfrac{\lambda}{r} q_{0,0} & K = 1 \end{cases} \\
q_{K,0} &= \begin{cases} \dfrac{\lambda}{m\mu}\left[q_{K-1,0} + q_{K-1,1}\right] & K > 1 \\ \dfrac{\lambda}{\mu} q_{0,0} & K = 1 \end{cases}
\end{aligned} \tag{6}
$$

To solve for $q_{0,0}$, we first solve for the normalized variables $q_{k,m}/q_{0,0}$ using Equation (1) to derive $q_{1,0}/q_{0,0}$, $q_{1,1}/q_{0,0}$, $q_{2,0}/q_{0,0}$. The terms $q_{k,1}/q_{0,0}$ and $q_{k+1,0}/q_{0,0}$ can be determined recursively from Equations (4) and (5). An algorithm can be developed to determine $q_{3,0}/q_{0,0}$ and substitute it to determine $q_{3,1}/q_{0,0}$, and then successively determine $q_{4,0}/q_{0,0}$, $q_{4,1}/q_{0,0}$, …, $q_{K,0}/q_{0,0}$ and $q_{K,1}/q_{0,0}$. Subsequently, $q_{0,0}$ can be found using the normalization condition

$$
p_0 = q_{0,0} = \frac{1}{\sum_{k=0}^{K}\left(\dfrac{q_{k,0}}{q_{0,0}} + \dfrac{q_{k,1}}{q_{0,0}}\right)} = \frac{1}{1 + \sum_{k=1}^{K}\left(\dfrac{q_{k,0}}{q_{0,0}} + \dfrac{q_{k,1}}{q_{0,0}}\right)}. \tag{7}
$$

Obtaining $q_{0,0}$ can be used to find all other state probabilities $\{q_{k,n}; 1 \le k \le K, n = 0,1\}$. Note that $q_{0,0}$ denotes the probability that the system is empty, i.e., p_0.

The mean system throughput γ is basically the departure rate, or equivalently the rate at which the requests finish being processed successfully by the compute instances, i.e.,

$$\gamma = \mu \sum_{k=1}^{K} q_{k,0}. \tag{8}$$

The probability P_{loss} is the loss or blocking probability. P_{loss} can be expressed as the probability of being in either state (K,0) or state (K,1), that is

$$P_{loss} = q_{K,0} + q_{K,1} .$$

The mean number of requests $\overline{K}$ in the system can be expressed as

$$\overline{K} = \sum_{k=0,n=0,1}^{K} k q_{k,n} = \sum_{k=1}^{K} k(q_{k,0} + q_{k,1}) .$$

Using Little's result, the mean time spent in the system by a request succeeding in entering the queue can be expressed as

$$W = \frac{\overline{K}}{\gamma} = \frac{1}{\gamma}\sum_{k=0}^{K-1} k(q_{k,0} + q_{k,1}) + \frac{K}{\gamma}(q_{K,0} + q_{K,1}) .$$

The CPU utilization (known also as the *carried or working load*) of each VM instance can be expressed as follows

$$U_{util} = \frac{\gamma}{m\mu} ,$$

where γ is expressed in Equation (8).

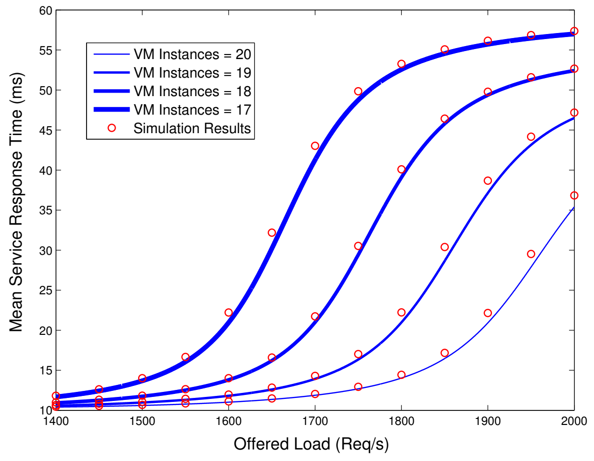
III. VERIFICATION

To verify the correctness of our analytical model, we have developed a discrete-event simulation of a finite queueing system with an LB and multiple servers taking into account the same assumptions as those in the analysis. The simulation was written in C language, and the code followed closely and carefully the guidelines given in [19]. We used the PMMLCG as our random number generator [19]. The simulation was automated to produce independent replications with different initial seeds that were ten million apart. During the simulation run, we checked for overlapping in the random number streams and ascertained that such a condition did not exist. The simulation was terminated when achieving a precision of no more than 10% of the mean with a confidence of 95%. We employed and implemented dynamically the *replication/deletion* approach for means discussed in [19]. We computed the length of the initial transient period using the MCR (Marginal Confidence Rule) heuristic developed by White [20]. Each replication run lasts for five times of the length of the initial transient period.

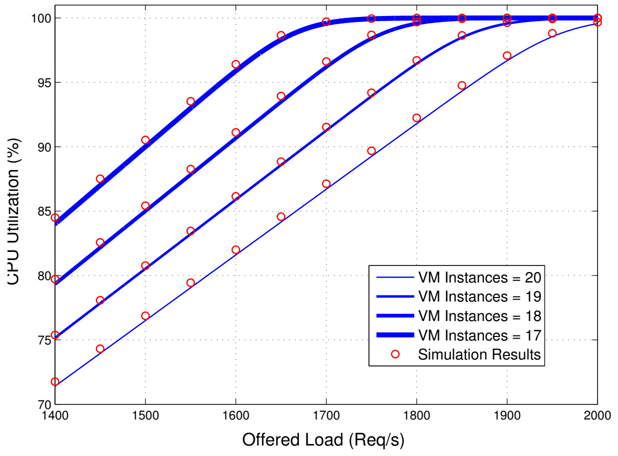
IV. NUMERICAL EXAMPLES

In this section, we report numerical results obtained by using both analytical model and simulation. The analytical curves were obtained by MATLAB implementation of the equations derived from the analytical models. The simulation results were obtained using a discrete-event simulation described in Section III. In Figure 4, the results obtained from simulation are represented by the red circles, whereas the curves represented by lines are those of analysis. The figure shows that both simulation and analysis results are in agreement, and thus implying that our analytical model is correct.For this numerical example, we use a typical web workload as reported in [12,21,22]. We fix the system size K to 100 requests. We fix $1/r = 0.2\,ms$ and $1/\mu = 10\,ms$. The mean service rate μ is realistic and consistent with the reported experimental rates in [12,21,22], and this service includes CPU processing in addition to any required disk I/O or database access.

To show how the model estimates the service time, and also to illustrate how the model can estimate the number of required VM instances (or compute nodes) needed to satisfy a given response time, we plot the mean system delay and CPU utilization with respect to offered load λ. Figure 4 exhibits the impact of varying the number of VM instances on the mean system latency and its corresponding CPU utilization. We focus on the area of interest, in which the latency starts increasing, that is, when the arrival rate λ is between 1400 and 2000 Req/s.



(a)



(b)

Figure 4. Impact of provisioning multiple VMs on response times and its corresponding CPU utilizations

As shown, and as expected, smaller latencies (and lower CPU utilizations) are exhibited with larger number of VM instances. Given a measured offered load, the minimum number of VM instances can be determined to satisfy a given response time. For example, if the SLA service response time to satisfy (excluding network delays) is 15 ms, and the expected offered load (set at the start for the elastic application or measured later by the LB) is 1500 Req/s, then the needed VM instances will be 17, as shown in Figure 4(a). However, if the offered load grows to 1600 Req/s, the needed VM instances will be 18, and so on. Figure 4(b) shows the corresponding CPU utilizations of a slightly over 90% for 17 and 18 VM instances at an offered load of 1500 and 1600 Req/s, respectively. In Amazon auto-scaling, a threshold of 85% (set arbitrary by the user) is typically recommended to trigger scaling out (i.e. adding more VM instances) so that a service response time can be met [4]. However, and as shown, this arbitrary threshold is not appropriate in determining the needed number of VM instances to guarantee a given service response time. As shown in Figure 4(b), with 85% CPU utilization, 20 VM instances are needed to maintain the response time below 15 ms. This leads to unnecessary over-provisioning and poor utilization of cloud resources, and thereby resulting in a higher system cost for the cloud customer. In conclusion, the VM instances should be sized based on the given response time

V. CONCLUSION

We have presented a Markovian analytical model to estimate the service response time for an elastic cloud application. The model estimates the service time based on the number of provisioned VM instances, the capacity of each VM instance, and the expected load. Simulation results show that our analytical model is correct. Numerical examples have been given to illustrate how the model can predict accurately the minimum number of VM instances to satisfy the response time. We have demonstrated that today's practice of using an arbitrary threshold for CPU utilization to auto-scale resources is not an accurate measure and can lead to high cost and poor utilization of resources. As a future study, we plan to validate our model experimentally by measuring the performance of popular elastic applications hosted on Amazon cloud, and also compare our experimental results with those of our analytical model. To date, no experimental results have been reported in the literature on gauging the processing capacity of VM instances (in Req/s) or on measuring the performance of elastic applications in terms of response time and required VM instances when subjected to different loads.

REFERENCES

[1] W. Iqbal, M. Dailey, D. Carrera, and P. Janecek, “Adaptive Resource Provisioning for Read Intensive Multi-tier Applications in the Cloud”, Journal of Future Generation Computer Systems, Eslevier Science, Vol. 27, No. 6, June 2011, pp. 871-879.

[2] H. Liu and S. Wee, “Web Server Farm in the Cloud: Performance Evaluation and Dynamic Architecture,” Proceedings of the 1st 2009 International Conference on Cloud Computing, Springer-Verlag, 2009, pp. 369-380.

[3] A. Azeez, “Auto-scaling Web Services on Amazon EC2”, 2012. Available at http://people.apache.org/~azeez/autoscaling-web-services-azeez.pdf

[4] Amazon Inc., “Amazon Web Services Auto Scaling,” 2012. Available at http://aws.amazon.com/autoscaling

[5] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. Lara, M. Brudno, M. Satyanarayanan, “SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing,” Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys'09, Nuremberg, Germany, March 2009, pp. 1-12.

[6] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser, “AppRAISE: Application-Level Performance Management in Virtualized Server Environments,” IEEE Transactions on Network and Service Management, Vol. 6, No. 4, December 2008, pp. 240-254.

[7] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, “Agile Dynamic Provisioning of Mult-tier Internet Applications,” ACM Transactions on Autonomous and Adaptive Systems, Vol. 3, 2008, pp. 1-39.

[8] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, A. Tantawi, “An Analytical Model for Multi-tier Internet Services and its Applications,” Proceedings of the 2005 ACM SIGMETRICS International Conference, Vol. 33, Alberta, Canada, pp. 291-302.

[9] H. Khazaei, J. Misic, and V. Misic, “Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queueing Systems,” IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 5, May 2012, pp. 936-943.

[10] S. Kikuchi and Y. Matsumoto, “Performance Modeling of Concurrent Live Migration Operations in Cloud Computing Systems using PRISM Probabilistic Model Checker,” Proceedings of the 4th IEEE International Conference on Cloud Computing, 2011, Melbourne, Australia, pp. 49-56.

[11] M. Firdhous, O. Ghazali, and S. Hassan, “Modeling of Cloud System using Erlang Formulas,” Proceedings of the 2011 7th Asia-Pacific Conference on Communications (APCC), Saba, Malaysia, October, 2011, pp. 411-416.

[12] K. Xiong and H. Perros, “Service Performance and Analysis in Cloud Computing,” Proceedings of the 2009 IEEE Congress on Services, July 2009, Los Angeles, Californian, pp. 693-700.

[13] W. Leland, M. Taqqu, W. Willinger, D. Wilson, "On the Self-Similar Nature of Ethernet Traffic", IEEE/ACM Transaction on Networking, vol. 2, no. 1, February 1994, pp. 1-15

[14] V. Paxson and S. Floyd, “Wide-Area Traffic: The Failure of Poisson Modeling,” IEEE/ACM Transactions on Networking, vol. 3, no. 3, June 1995, pp. 226-244

[15] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, “Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level,” Proceedings of ACM SIGCOMM, Cambridge, Massachusetts, August 1995, pp. 100-113.

[16] K. Salah, K. Elbadawi, and R. Boutaba, “Performance Modeling and Analysis of Network Firewalls” *IEEE Transactions on Network and Service Management*, Elsevier Science, vol. 9, no. 1, March 2012, pp. 12-21

[17] R. D. Van Der Mei, R. Hariharan, and P. K. Reeser, "Web Server Performance Modeling," Journal of Telecommunication Systems, vol. 16, no. 3-4, 2001, pp. 361-378.

[18] K. M. Chandy and C. H. Sauer, “Approximate methods for analyzing queueing network models of computing systems,” *Journal of ACM Computing Surveys*, vol. 10, no. 3, September 1978, pp. 281-317.

[19] A. Law and W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 2nd Edition, 1991.

[20] J. White, “An Effective Truncation Heuristic for Bias Reduction in Simulation Output,” *Simulation Journal*, vol. 69, no. 6, December 1997, pp. 323-334

[21] J. Bi, Z. Zhu, R. Tian, and Q. Wang, “Dynamic Provisioning for Virtualized Multi-tier Applications in Cloud Data Center,” Proceedings of the 2010 IEEE International Conference on Cloud Computing, Miami, Florida, July 2010, pp. 370-377.

[22] J. Dejun, G. Pierre, and C.-H. Chi, “EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications,” Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, November 2009.