Using Simulations to Assess the Stability and Capacity of Cloud Computing Systems

Jingsong Wang
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
wang82@email.sc.edu

Michael N. Huhns
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
huhns@cec.sc.edu

ABSTRACT

For applications hosted in a cloud computing system, where there are many servers to handle incoming invocations of the application, the assessment of stability of the cloud is very important for both the planning of new applications and the expansion of existing applications. However, a general assessment is always hard to achieve as there are still no standard definitions of techniques used in cloud computing and an actual cloud computing system could be very large and complex. This paper presents a simulation for a cloud computing environment. It enables an assessment of the cloud's logical stability under various configurations without performing experiments on the actual cloud environment. The correctness of the simulation is verified by the theoretical calculation results of the well known M/M/1 queuing system.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services, Data sharing*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems

General Terms

Design, Experimentation, Measurement, Performance, Reliability

Keywords

Cloud Computing, M/M/1 Queuing System

1. INTRODUCTION

1.1 Cloud Computing System

Although there is still no standard definition for the concept of *cloud computing*, regarded as the next natural step in the evolution of on-demand information technology services and products [10], cloud computing is beginning to have a significant impact on both individual end users and enterprises, and is attracting increasing research and development from academia and industry. We choose herein the definition from [9], as it includes a comprehensive analysis of the features of cloud computing. *"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs."*

As described in the 2009 Horizon Report [7], which put cloud computing as one of six emerging technologies to watch, *"The emergence of large-scale 'data farms' - large clusters of networked servers - is bringing huge quantities of processing power and storage capacity within easy reach. Inexpensive, simple solutions to offsite storage, multi-user application scaling, hosting, and multi-processor computing are opening the door to wholly different ways of thinking about computers, software, and files."*

It is a style of computing in which massively scalable IT-enabled capabilities are provided "as a service" to multiple customers. Unlike previous IT licensing models, however, these services are typically billed to customers on a consumption basis, thereby converting the traditional capital expenditure model common in data centers today to an operational expenditure model. The services provided by cloud computing environments are based strongly on the use of virtualization of various types of computing technologies, but the common theme is reliance on the Internet to satisfy the computing needs of users. The types of computing technologies being virtualized lead to the following four categories of cloud services: Basic Computational Services, IaaS, PaaS, and SaaS.

But just like any other service-based activities, no matter what category the service is, it mainly involves two actors: the service requester and the service provider. From the service requester's view, a cloud computing system provides a computing utility available on demand. From the service provider's view, at the system level, cloud computing contains a robust allocation system that rationally distributes incoming requests to specific computer servers in the cloud to handle the request. Notice that, being a rapidly evolving research topic, cloud computing still has no agreed-upon

ACMSE'10 April 15-17, 2010, Oxford, MS, USA.

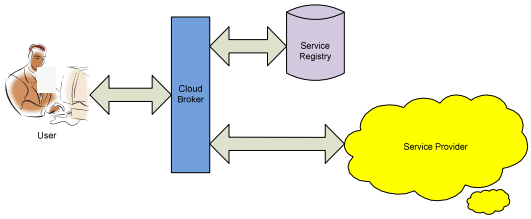

Figure 1: The basic architecture of a cloud computing system.

standards, so here we make use of a very high-level abstraction for the service provider side. We discuss more about its logical stability assessment based on system configurations in the coming sections. An actual cloud computing system will certainly include additional components, such as service brokers and cloud federations. The former negotiates relationships between service requesters and service providers of a cloud, while the latter is very necessary for social network sites whose components could be hosted by different cloud computing providers in order to optimally serve customers at various geographical locations [3]. These techniques play the same significant roles for the whole system's performance as those servers that process incoming requests. However, they are not the focus of this paper. Figure 1 shows a basic architecture of a cloud computing system. Our analytical model presented in section two can be simply considered as an abstraction built over the service provider, which always contains a large network of cloud servers executing the actual computing tasks.

There are many advantages service users can take from cloud computing. Six of the main benefits are summarized in [2]: (1) reduced cost, (2) increased storage, (3) highly automated, (4) flexibility, (5) more mobility, (6) allows IT to shift focus. Meanwhile, the large scale application of cloud computing brings many issues and challenges. One major challenge of moving applications to the cloud is the need to master multiple languages and operating environments. Also, cloud computing raises questions about privacy, security, and reliability [6]. Many research issues are presented in [10]. For example, the open challenges related to cloud provenance data include how to collect provenance information in a standardized and seamless way with minimal overhead, how to store it in a permanent way so that one can come back to it at anytime, and how to present it to the user in a logical manner. Among the problems, system stability is always one of the top concerns and thus the technical issue we address here is an assessment of a system's stability. The resulting experimental outputs can also be used for capacity planning in the design stage of a cloud system development.

1.2 Stability Assessment

Stability for a cloud computing system, just like for any other system, is of utmost importance. Stability assessment is needed for the whole system life time, from the initial design and implementation to actual operation, management, and extension. Customers would not tolerate a slow response from a cloud application, because they are paying for its use on demand. Service providers need to provide quality systems that can make full use of existing resources, while serving every request in time to attract more customers. The interruption of Amazon Web Services and Gmail's unavailability for only a couple of hours recently led to a number of complaints from users who had trouble accessing their accounts, and these cases have raised questions and worries about the stability of cloud computing systems.

For a large cloud computing system, there are numerous factors that essentially govern its stability, such as the condition of the software and hardware supporting the system. One exception in an operating system or a brief malfunction of a network card may put the whole system into an unstable state. This kind of stability problem caused by the faults or loss of equipment can be defined as a physical stability problem, which is not the one we address in this paper. We pay more attention to a system's logical stability, i.e., we focus on the stability issues caused by various system configurations in terms of three basic metrics: the arrival rate of requests, the number of servers in the cloud, and the computing capacity of each server. We then consider how to use simulation to assess stability in terms of the metrics. Also as stated above, the assessment is restricted to the system level at the service provider side, instead of the global system containing every cloud component, and is still theoretical as it is based on high-level abstractions.

For a cloud computing system, whose computing capacity is fixed, stability is crucial, because bad effects can accumulate. This generally happens when the current requested computing capacity exceeds the available service capacity. One direct result is that these requests cannot be served immediately and have to wait. If the number of requests keeps increasing at the same or even higher rate, the cloud computing system then cannot handle incoming requests in a timely manner and the number of requests waiting or never served will grow. We say that the system in this situation is not stable.

This situation worsens when waiting requests are resent. For an existing system, we can assess its stability by looking at its log files and monitoring its performance. The assessment accuracy depends on the data collected and is limited by the available resources. In fact, monitoring cloud systems is also an active research topic. The enormous size of a cloud data center and its large number of nodes require a robust monitoring system to actively react to failures [11]. It would be better to predict the stability condition in advance than react when failures really occur. However, in the planning period of building or expanding a large cloud system, generally we have no physical ones to observe or we cannot afford to use the real platform to assess its stability. In these cases, we must resort to simulation. Simulation makes possible repeatable results and exploitation of various scenarios in a light-weight environment. A good simulation program for stability assessment should reflect the potential problems or help in finding an optimum system configuration before the designed system is constructed and deployed.

The rest of this paper is organized as follows. Section two describes the system definition and the simulation design. Section three shows the experimental results, including the results for verification using M/M/1 queuing theory. We conclude our work in section four with a discussion of related and future work.

2. SIMULATION DESIGN

2.1 System Definition

Although there are many different cloud computing sys-

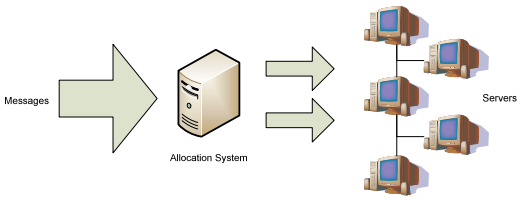

Figure 2: A simplified message processing process of a cloud computing service provider.

tems, which may employ various standards and techniques, the basic architectures and processing procedures of actual service providers do not vary very much. Hence, they can be reasonably modeled as described in [5] for theoretical analysis purposes, as follows: we consider a group of computer servers arranged to cooperate in providing cloud computing services to a stream of randomly arriving messages, standing for requests/programs to be executed. Each server is assumed to have the same maximum possible computing rate in Floating Point Operations per Second (FLOPS). Messages can be of different types, and each type of message has a specified service capacity and requires a specified mean execution time. An arriving message requiring specific computer capacity will be allocated to a server and begin executing immediately, if the required cloud capacity is available. Otherwise, it will be placed in a queue and wait for a given time; it will then retry for service. One message can only reside on a single cloud element (a computer server) at a time. When the arrival rate of messages surpasses the service capacity of the cloud computing system, the queue will tend to grow indefinitely, and the system is defined to be unstable or saturated. Figure 2 shows how messages presented above are handled at the service provider side.

2.2 Simulation Design

Based on the theoretical model of the previous section, we design a program to simulate its performance. A few necessary assumptions are made as follows. We assume that all the servers in the cloud are identical, i.e., have the same computing capacity. In addition, we assume that there are at most two types of messages that arrive according to a Poisson process with given rates. Their service times are assumed to follow an exponential distribution with given means. We also assume that the policy of choosing servers is without replacement, i.e., an arriving message first randomly chooses one server to request processing; if the request is rejected then the message requests service from the remaining servers at random; when there is no server available to try, the rejected message is placed into the queue. Our simulation mainly measures the size of the queue of messages that forms as processing proceeds by the servers.[1] Our detailed descriptions about system structure, input and output design, and flow of control are online at `http://www.cse.sc.edu/~wang82/doc/ccs2009`.

3. EXPERIMENTAL RESULTS

[1]An actual cloud computing system may have many more types of messages that arrive according to other distributions, and more complex policies can exist not only for choosing a server, but also for deciding how to serve a message on a specific server.

Experiments were conducted on a desktop computer to verify the correctness of the simulator.

3.1 Correctness Verification

Because the simulation program provides the flexibility of setting values of most parameters, we can let it approximately simulate the run of a standard M/M/1 queuing system by fixing the server number to be one. Theoretically, there are well founded mathematical solutions for the calculation of measurements like the average queue length, the average number of requests in the system, the average time spent in the system, the average time spent in the queue, and so on, with respect to the mean arrival rate and the mean service rate in the M/M/1 system. As a result, M/M/1 is well suited for validating simulation results.

3.1.1 M/M/1 Queuing System

The M/M/1 system [1] is made of:

- Arrivals are a Poisson process (λ);
- Service time is exponentially distributed ($1/\mu$);
- There is one server;
- The length of queue in which arriving requests wait before being served is infinite;
- The population of requests available to join the system is infinite.

We define $\rho = \lambda/\mu$. Then we can use the following equation to find the number of requests in the queue:

$$N = \rho^2/(1-\rho)$$

3.1.2 M/M/1 Simulation Setting and Results

To simulate an M/M/1 queuing system, we have parameter values set as follows: we set Server Number to be 1, the message Arrival Rate to be 1.0, and message Retry Time to be 0.5. Particularly, we set Capacity Per Server to be 5. Requested Capacity is 5 and Service Time is 0.75 for both two types of messages. So the system has only one type of messages, and at any time, only one message could be served in the server, and the actual service rate μ could be calculated to be:

$$\mu = 1/0.75 = 20/15 \approx 1.3333$$

Then such a system is very close to an M/M/1 queuing system, except that each message has a retry time. In a standard M/M/1 system, the retry time for a queued message should be zero, i.e., as soon as a server is available, the message will be sent to that server immediately.

When we have parameters set this way, we can run a few simulations and use actual outputs to compute actual values for ρ and then N, the theoretical average queue length in a long run. In the end, we can compare the computed N with the actual output average queue length. The degree of closeness will tell whether our system really handles messages in a proper way, or whether the program simulates a queuing system correctly.

At first, we run the simulation program ten times with expected arrival rate $\lambda = 1.0$ and retry time to be 0.5. Table 1 shows the actual arrival rate and the computed average queue length, and the actual average queue length. We use λ to represent message arrival rate, μ to represent service

Table 1: Expected arrival rate $\lambda = 1.0$, retry time = 0.5, in 1 server case (5000 seconds)

Number	Actual Input		Theoretical Computation		Actual Output		Difference	
	λ	μ	ρ	N	L_1	L_2	$L_1 - N$	$L_2 - N$
1	0.9936	1.3416	0.7406	2.1146	3.2008	3.1511	1.0862	1.0365
2	1.0009	1.3353	0.7496	2.2436	3.8936	3.8366	1.6500	1.5930
3	0.9924	1.3203	0.7516	2.2749	3.7333	3.6875	1.4584	1.4126
4	0.9948	1.3237	0.7515	2.2731	3.7957	3.7260	1.5226	1.4529
5	1.0042	1.3327	0.7535	2.3034	3.3631	3.3020	1.0597	0.9986
6	1.0184	1.3436	0.7580	2.3736	3.1267	3.0698	0.7531	0.6962
7	1.0344	1.3429	0.7703	2.5827	3.6725	3.6215	1.0898	1.0388
8	0.9952	1.3241	0.7516	2.2742	3.0875	3.0299	0.8133	0.7557
9	0.9872	1.3122	0.7523	2.2852	3.1084	3.0671	0.8232	0.7819
10	0.9982	1.3488	0.7401	2.1071	2.9316	2.8718	0.8245	0.7647
							1.1081	1.0531
Mean	1.00193	1.33252	0.7519	2.2788	3.3913	3.33633	1.1125	1.0575

Table 2: Expected arrival rate $\lambda = 1.0$, retry time = 0.05, in 1 server case (5000 seconds)

Number	Actual Input		Theoretical Computation		Actual Output		Difference	
	λ	μ	ρ	N	L_1	L_2	$L_1 - N$	$L_2 - N$
1	1.0108	1.3347	0.7573	2.3634	2.4434	2.5117	0.08	0.1483
2	1.0110	1.3121	0.7705	2.5872	2.3938	2.4455	-0.193	-0.142
3	0.9874	1.3168	0.7498	2.2477	2.0524	2.1183	-0.195	-0.129
4	0.9938	1.3642	0.7285	1.9546	1.7877	1.8680	-0.167	-0.087
5	1.0046	1.3455	0.7466	2.2003	1.6393	1.7242	-0.561	-0.476
6	0.9924	1.3247	0.7492	2.2373	2.6178	2.6802	0.3805	0.4429
7	1.0126	1.3521	0.7489	2.2337	3.0532	3.1215	0.8195	0.8878
8	0.9844	1.3284	0.7410	2.1206	2.4230	2.4973	0.3024	0.3767
9	1.0070	1.3402	0.7514	2.2708	2.3774	2.4401	0.1066	0.1693
10	0.9998	1.3478	0.7418	2.1312	2.5778	2.6376	0.4466	0.5064
							0.1019	0.1698
Mean	1.00038	1.33665	0.7484	2.2265	2.3366	2.40444	0.1101	0.1779

rate, $L1$ and $L2$ to represent the average queue lengths from two different ways of calculation, one by output per second, and the other is based on integral. $L1 - N$ and $L2 - N$ show the differences.

From Table 1, we can see that the actual arrival rate 1.00193 and service rate 1.33252 is very close to the values we set (expected) $\lambda = 1.0$ and $\mu = 1.3333$, and the output average queue length $L1$ and $L2$ are just a little more than the theoretically computed N values. The difference is about 1 message more. Most importantly, in ten random runs, the system stably produces values which are in a fixed range that is close to the theoretical computation results. Then we use the same parameter values except a different retry time. We reduce the retry time from 0.5 to be 0.05. We still randomly run ten times the simulation program and get ten sets of outputs shown in Table 2. From Table 2, it is very obvious that the difference between the actual output $L1$ and $L2$ and N is much smaller. It decreases from about 1 message to about 0.1 message, a very close value, when having retry time reduced from 0.5 to 0.05. To further verifying the correctness of simulation, we change the expected arrival rate to be 0.5 and all the other parameter values are kept same as before, i.e. still the same expected service rate. Table 3 shows the actual outputs, where the average difference can reach 0.006, which is very good.

When the arrival rate is higher than the system service rate, the queue length should apparently increase. We make some runs with such settings and even longer simulation time, 10k seconds, and we can see the increase change of queue length from Table 4. From Figure 3, 4, and 5, we can see the queue length increases rapidly as the time value increases. Especially, the higher the ratio of λ/μ, the steeper is the shape.

Table 3: Expected arrival rate $\lambda = 0.5$, retry time = 0.05, in 1 server case (5000 seconds)

Number	Actual Input		Theoretical Computation		Actual Output		Difference	
	λ	μ	ρ	N	L_1	L_2	$L_1 - N$	$L_2 - N$
1	0.5082	1.2865	0.395	0.2579	0.2235	0.2839	-0.034	0.026
2	0.5190	1.3369	0.3882	0.2463	0.2203	0.2769	-0.026	0.0306
3	0.4936	1.3434	0.3674	0.2134	0.2183	0.2749	0.0049	0.0615
4	0.4906	1.3062	0.3756	0.2259	0.2707	0.3303	0.0448	0.1044
5	0.4864	1.3038	0.3731	0.2220	0.2387	0.2967	0.0167	0.0747
6	0.5058	1.3197	0.3833	0.2382	0.2687	0.3283	0.0305	0.0901
7	0.5002	1.3436	0.3723	0.2208	0.2267	0.2723	0.0059	0.0515
8	0.4864	1.3310	0.3654	0.2105	0.1943	0.2454	-0.016	0.0349
9	0.4996	1.3127	0.3806	0.2338	0.2559	0.3152	0.0221	0.0814
10	0.4990	1.3086	0.3813	0.235	0.2471	0.2986	0.0121	0.0636
							0.0060	0.0619
Mean	0.49888	1.31924	0.3813	0.235	0.2364	0.29225	0.0014	0.0572

Table 4: $\lambda > \mu$ in 1 server case (10k seconds)

Number	λ	μ	λ/μ	L_1	L_2
1	1.3490	1.3449	1.0030	156.5855	156.5745
2	1.3648	1.3151	1.0378	380.1344	380.1154
3	1.3698	1.3366	1.0248	332.4943	332.4885

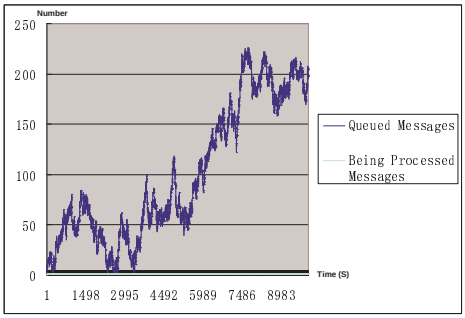

Figure 3: $\lambda = 1.3490$ (#1 in Table 4), in 10k seconds.

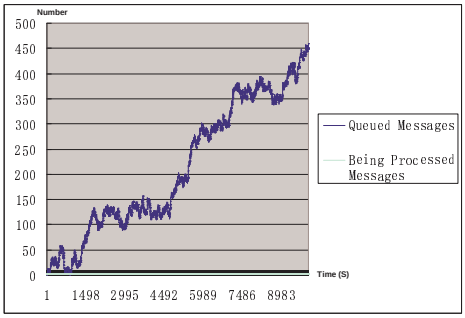

Figure 4: $\lambda = 1.3648$ (#2 in Table 4), in 10k seconds.

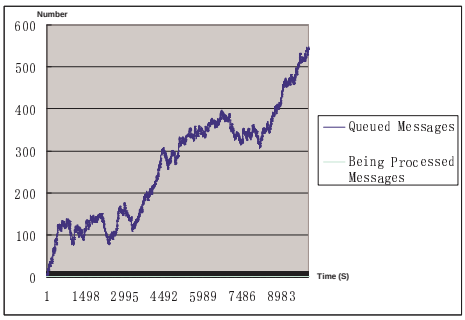

Figure 5: $\lambda = 1.3698$ (#3 in Table 4), in 10k seconds.

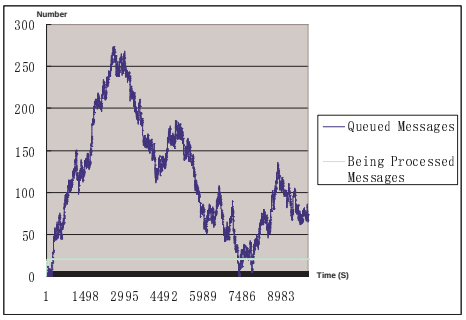

Figure 6: $\lambda = 1.3519$, $\mu = 1.3456$, in 10k seconds.

Table 5: Parameter Values for Message Types

Message	Type 1	Type 2
Capacity Requested (FLOPS)	5	10
Mean Execution/Service Time (Seconds)	15	12
Probability of Message	0.5	0.5

3.1.3 *Service Rate Discussion*

If we reset some of the parameter values, for example, Capacity Per Server to be 100, Requested Capacity to be 5, and Service Time to be 15 for both two types of messages, i.e., the system still has only one type of messages, but at any time, more than one message (in fact it can be at most 20 messages) could be served in the server, then it seems that the actual service rate μ could be calculated to be:

$$\mu = 20/15 = 1/0.75 \approx 1.3333$$

which is equal to the service rate we have in the system we simulated in the previous section. But the output of actual runs shows that the actual service rate of this system is slightly higher than we thought and calculated. We choose one sample with input arrival rate 1.34, actual arrival rate 1.3519, and the calculated service rate 1.3456, running in 10k seconds. Figure 6 shows the queue length fluctuation in the run where we do not find the same way of increase shown in Figure 3, 4, and 5, even if we have $1.3519 > 1.3456$, whose ratio is $\lambda/\mu = 1.0047$, which is even higher than #1 sample in Table 4.

3.2 Simulations for More Servers

Now we return to the more complex case. We simulate a system with 500 servers and 2 types of messages as defined in Table 5. Each server has the same capacity: 100 FLOPs. We do three simulation replications.

Because there are two types of messages, we do not have a simple way as before to know the possible service rate, but we can now approximate its value from the simulation outputs. We have found that the system is in a stable state when the arrival rate is around 510. The actual arrival rate and outputs are shown in Table 6. Figure 7 shows that the queue length is very small when λ is smaller than 510 (corresponding to #4 in Table 6). When λ is close to 510, we can see that the queue length fluctuates very much, as shown in Figure 8, 9, and 10 (corresponding to #1, #2, #3 in Table 6 respectively), but the system is still in a stable state. When we choose a higher value for the expected arrival rate, 515, we can see apparently that the queue length goes straightly up as the time increases, even within a shorter 5000 seconds, in Figure 11 (corresponding to #5 in Table 6).

Table 6: Two message types in 500 server case in 10k and 5k seconds separately

Time	Number	λ	L_1	L_2
10000s	1	509.8101	145.3111	145.2435
	2	510.1773	143.6217	143.6174
	3	510.1916	169.2893	169.2670
	4	499.9239	2.2004	2.1804
5000s	5	514.5486	5384.8341	5384.7288

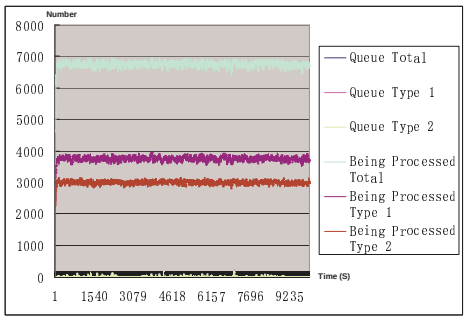

Figure 7: $\lambda = 499.9239$ (#4 in Table7), 500 servers, 10k seconds.

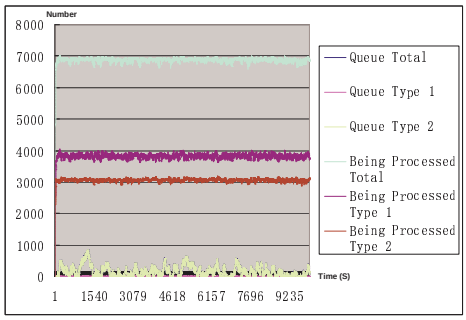

Figure 8: $\lambda = 509.8101$ (#1 in Table 6), 500 servers, 10k seconds.

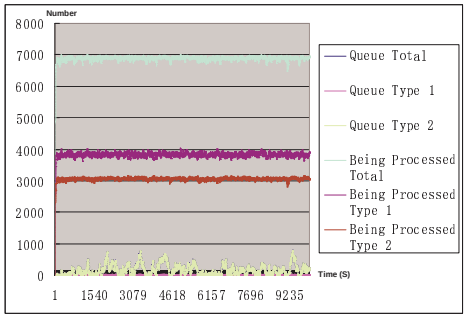

Figure 9: $\lambda = 510.1773$ (#2 in Table 6), 500 servers, 10k seconds.

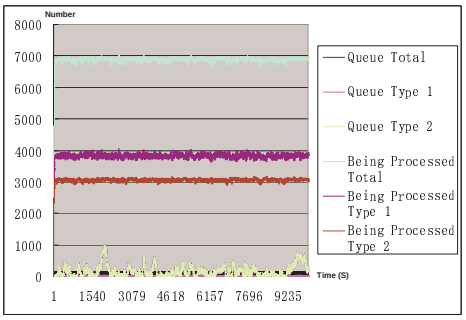

Figure 10: $\lambda = 510.1916$ (#3 in Table 6), 500 servers, 10k seconds.

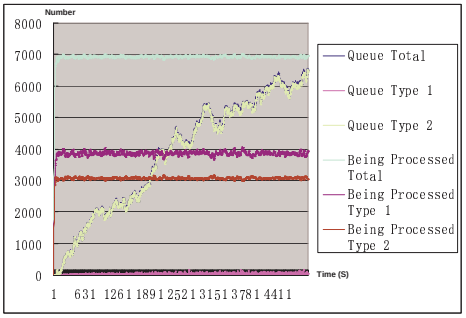

Figure 11: $\lambda = 514.5486$ (#5 in Table 6), 500 servers, 5k seconds.

4. CONCLUSION AND DISCUSSION

This paper presents a simulation program for a stability assessment of a cloud computing system. The stability assessment can also help us find the actual service rate. The simulation program has provided flexibility in setting values of critical parameters. The only restricted parameter is the number of message types(two currently). But from the way of design we can see that it is not hard at all to extend the program to accommodate more types of messages. Unlike other computing techniques in the distributed environment, such as Grid computing for which several simulators have been proposed, including GanSim [4] and SimGrid [8], we have not found many efforts for the cloud computing paradigm, especially for its stability analysis. [3] has made a good attempt. However it is based on the analysis of a global system and tries to simulate most aspects of a cloud computing system. Instead of stability, its experiments focus more on scalability issues of such a comprehensive system and quantification of its performance using some basic configuration, which lacks natural variety. Also, as emphasized by [3] itself, cloud computing is still a rapidly evolving research area and there is a severe lack of defined standards. [3]'s simulation results have been restricted by their own architectures and standards, and have reduced its generality for other systems. In addition, in their work they provide no way of verifying the simulator, which is critical for most simulations. In our work, we focus on a special issue that is smaller but applicable to various cloud computing systems, because the basic system level architecture we modeled will not vary much. We have used the well-known theory of M/M/1 queuing systems to verify the correctness of our simulation. There are a number of avenues for future exploration via the simulation of cloud computing environments, as follows: 1) Improve the simulation program to accept more types of messages. This improvement should be very easily achieved. 2) Study the queued possibility of messages regarding their types. We have seen from the experimental results that the type of messages requiring more capacity will have higher probability to be queued, even if they need less execution time. We can study the correlations between these two features to find the threshold that has deterministic influence. 3) Research the influence of system architecture to its actual service rates. As we noticed before, for one server system, the ratio of the required capacity of a single message to the total capacity in the server, i.e., the number of messages a server can process at a moment, has affected the system's actual service rate. We need to investigate whether it matters in a multi-server case.

5. ACKNOWLEDGEMENTS

The authors wish to thank Dr. William Lewis, who inspired and guided the work described in this paper.

6. REFERENCES

[1] M/m/1 model. Available at http://en.wikipedia.org/wiki/M/M/1 model.

[2] Six benefits of cloud computing. Available at http://web2.sys-con.com/node/640237.

[3] R. Buyya, R. Ranjan, and R. N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *Proceedings of the 7th High Performance Computing and Simulation (HPCS 2009) Conference*, pages 21–24, Leipzig, Germany, July 2009.

[4] C. L. Dumitrescu and I. Foster. Gangsim: A simulator for grid scheduling studies. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 1151–1158, Washington, DC, USA, May 2005.

[5] D. P. Gaver, P. A. Jacobs, and W. C. Lewis. Models to assess stability of cloud computing, January 2009. To be published.

[6] B. Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, July 2008.

[7] L. Johnson, A. Levine, and R. Smith. The 2009 horizon report, 2009. Austin, Texas, The New Media Consortium.

[8] A. Legrand, L. Marchal, and H.Casanova. Scheduling distributed applications: the simgrid simulation framework. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 138–145, May 2003.

[9] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Toward a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2009.

[10] M. A. Vouk. Cloud computing : Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31–40, June 2008.

[11] L. Youseff, M. Butrico, and D. D. Silva. Towards a unified ontology of cloud computing. In *Grid Computing Environments Workshop (GCE08)*, pages 1–10, November 2008.