ISSN:0975-9646

A Controller Based Approach for Web Services Virtualized Instance Allocation

Sandesh Tripathi , S Q Abbas , Rizwan Beg

[1,2,3]*CSE Department, Integral university, Lucknow*

***Abstract*— Few Service providers provide compute intensive and data intensive services over web platform; where in applications can be deployed on demand. These service providers usually employ machine virtualization for providing cost effective solution. At the time of infrastructure purchase, one may opt for a particular instance, assuming that this will satisfy the computational needs. Whereas considering the case when no of service request increases, the more robust instance may be needed.**

In this paper, we present a mechanism which provides optimal instance allocation satisfying the computational needs. Results show that this approach utilizes the infrastructure in more optimal manner.

***Keywords*— Web Service, Allocating Services, Controller**

I. INTRODUCTION

Amazon web services (AWS) is a collection of remote computing services (also called web services) that together make a cloud computing environment. Few famous services are Amazon Elastic Compute Cloud (EC2) ,Amazon Simple DB, Amazon Simple Queue Service(SQS), Amazon Simple Storage service (S3) and many more. Launched in July 2002, Amazon Web Services provides on line services for the web sites or client side services. The services are not directly exposed to the end user but instead offer functionality that developers can use.AWS offerings are accessed over HTTP, using Representational State Transfer (REST) and SOAP protocols. These services are billed as per usages, but the usages are measured for billing varies from service to service. AWS allows obtaining a configurable capacity with minimal friction. It allows scaling the capacity quickly up and down. Hence changes the economics involved for computing and allow one to pay only for the capacity of actual use.

In this approach analytic performance models are combined with QCAC (QoS Computation and Configuration algorithm) , to design a controller mechanism, which runs periodically to determine the best possible instance type satisfying QoS parameters under present load, and choose the same. Such systems will encourage users to use controller based systems. Due to the dynamic instance shifting such systems are easily scalable.

II. UNDERSTANDING INSTANCE BASED SERVICES

A. Need of Utility Computing

Utility computing is a service provisioning model in which a service provider makes computing resources and Infrastructure management available to the customer as needed, and charges for specific usages rather than flat rate. Utility computing promises to cut costs. AWS was the first, who offered this model. Among few examples are, A) 6 waves limited a leading international publisher and developer of gaming applications on the Facebook platform, uses Amazon EC2 and Amazon S3 to host its social games with an audience of more than 50 million players per month, B) 99designs' massive design marketplace has received over 3.1 Million unique design submissions from over 53,000 designers around the world and runs entirely on AWS, C) YouOS uses S3 to allow 60,000 customers to store 12 GB of storage for its collaborative web operating system, are among few examples. Amazon says its services are designed to be scalable, fast, secure and simple.

B. Virtualization used in AWS

Xen Virtualization: Xen is an open source x86 virtual machine monitor which can create multiple virtual machines on a physical machine. Each virtual machine runs an instance of an operating system. A scheduler is running in the Xen hypervisor to schedule virtual machines on the processors. The original Xen implementation schedules virtual machines according to borrowed virtual time (BVT) algorithm [11]. For network virtualization, Xen only allows a special privileged virtual machine called driver domain, or domain 0 to directly control the network devices. All the other virtual machines (called guest domains) have to communicate through the driver domain to access the physical network devices. For this the driver domain has a set of drivers to control the physical network devices, and a set of back end interfaces to communicate with guest domains. The back end interfaces and the physical drivers are connected by a software bridge inside the kernel of the driver domain. Each guest domain has a customized virtual interface driver to communicate with a back end interface in the driver domain. All packets are sent from guest domain will be sent to the driver domain through the virtual interfaces and then sent into the network. All the packets which are destined to a guest domain will be received

by the driver domain first, and then transferred to the guest domain.

III. Related Work

Few studies have evaluated the performance of cloud services, where as little work has been done in the area of dynamic instance allocation. Guohui [12], studied the impact of virtualization on network performance of Amazon EC2 data centre. They presented a measurement study to characterize the impact of virtualization on networking performance of the Amazon Elastic Cloud Computing data centre. They measured the processor sharing, packet delay, TCP/UDP throughput and packet loss among Amazon EC2 virtual machines. Simson L. Garfinkel [20] evaluated and fined out in their study that Amazon Services have great promise but they lack service layer agreement. E. Walker[12] benchmarked Amazon EC2 for high performance scientific computing. The author evaluated how Amazon EC2 can be used for scientific computing. Our work signifies how dynamic allocation can be done for instances in agreement to QoS.

IV. Estimation of QoS

This metric combines the relative deviations of the average response time , average throughput, and probability of rejection with respect to the desired goals.

The relative deviation $\Delta QoS_{Relative}$ of the average response time is defined as

$$\Delta QoS_{Relative} = \frac{ART_{max} - ART_{measured}}{\max(ART_{max}, ART_{measured})} \quad (1)$$

ART_{max} is the maximum average response time, which can be tolerated and $ART_{measured}$ is the measured response time.

The properties of the above definition can be given as:

- $\Delta QoS_{Relative} = 0$ if the response time exactly meets its SLA i.e., $ART_{measured} = ART_{max}$.
- $\Delta QoS_{Relative} > 0$ if the response time exceeds its SLA i.e., $ART_{measured} < ART_{max}$. Given that the measured response time $ART_{measured}$ is at least equal to the sum $\sum_{i=1}^{K} D_i$ of the service demands D_i for all K resources [2], then by using eq. 1 ,it can be concluded that $\Delta QoS_{Relative} \leq 1 - (\sum_{i=1}^{K} D_i)/ ART_{max} < 1$.
- $\Delta QoS_{Relative} < 0$ if the response time does not meet its SLA i.e., $ART_{measured} > ART_{max}$. Then from eq. 1 it follows that $-1 < -(1 - ART_{max}/ART_{measured}) \leq \Delta QoS_{Relative}$

Taking an example if the measured response time is 6 sec and the maximum response time is 9 sec, then $\Delta QoS_{Relative} = \frac{6-9}{9} = -.33$ indicating that there is a 33% loss with respect to the measured response time. For meeting the SLA it would be necessary to cut down 33% of the measured response time to meet the SLA.

The relative deviation ΔQoS_p of the probability of rejection is similarly as $\Delta QoS_{Relative}$.

$$\Delta QoS_p = \frac{P_{max} - P_{measured}}{\max(P_{max}, P_{measured})} \quad (2)$$

Where P_{max} is the maximum probability of rejection tolerated and $P_{measured}$ is the measured probability of rejection.

The properties of the above definition can be given as:

- $\Delta QoS_p = 0$ if the probability of rejection exactly meets its SLA i.e., $P_{measured} = P_{max}$.
- $0 < \Delta QoS_p \leq 1$ if the probability of rejection exceeds its SLA.
- $1 \leq \Delta QoS_p < 0$ if the probability of rejection does not meet its SLA i.e., $P_{measured} > P_{max}$.

The relative deviation of the average throughput is defined as

$$\Delta QoS_{TP} = \frac{TP_{measured} - TP_{min}}{\max(TP_{measured}, TP_{min})} \quad (3)$$

Where $TP_{measured}$ is the measured throughput, $TP_{min} = (\lambda , TP_{min})$ is the minimal value between the arrival rate λ and the minimum required throughput TP_{min}.

The properties of the above definition can be given as:

- $\Delta QoS_{TP} = 0$ if the throughput meet its SLA i.e., $TP_{measured} = TP_{min}$.
- $0 < \Delta QoS_{TP} < 1$ if the throughput exceeds its SLA i.e., $TP_{measured} > TP_{min}$.
- $-1 < \Delta QoS_{TP} < 0$ if the throughput does not meet its SLA i.e., ., $TP_{measured} < TP_{min}$.

A single metric QoS can now be defined as a weighted sum of the three QoS deviations defined above. Thus,

Qos = $w_{relative} \times \Delta QoS_{Relative} + w_p \times \Delta QoS_p + w_{TP} \times \Delta QoS_{TP}$

Where = $w_{relative}, w_{TP},$ are weights, in the interval [0,1], determined by the management, to indicate the relative importance of response time, throughput and probability of rejection. QoS is a dimension less number between -1 and 1. If all three metrics meet or exceed their SLA, QoS ≥0. If QoS <1 , then at least one of the metrics does not meet its SLA.

V. System Description

The computer system consists of a multithreaded server that receives requests at a rate of λ requests per second. The system has m threads and the number of threads and the number of requests that can be in the system either waiting for a thread or using a thread equal to n (n ≥ m). Thus, requests

that arrive and find n requests in the system are rejected. When a thread is executing a request, it is using physical resource such as CPU or disk. So, the response time of a request can be broken into the following components: waiting for a thread (i.e software contention), waiting for a physical resource, and using a physical resource.

. The Approach for this self configuring web service compute cloud is assisted by the configuration controller mechanism that monitors the AWS Cloud, monitors various resources utilizing content, execute the QCAC (QoS Computation and Configuration Analysis Component) at regular interval, computes the best configuration for AWCC (Amazon web service Compute Cloud). As a result of this controller mechanism running continuously at regular intervals generates the reconfiguration commands to instruct the AWS to adapt the best configuration under present circumstances also satisfying the QoS.

The Architecture of the configuration controller mechanism has three main components: Request Capacity Analysing Component known as RCAC, Resource Utilization Analysing Component known as RUAC, and QoS Computation and Configuration Analysis Component known as QCAC. RCAC analyses the number of arriving requests from different service requesting computers, and computes parameters like the load intensity, arrival rate of requests and uses these statistical parameters for forecasting the load conditions in next interval. This data is used by QCAC as the input parameters of the Queuing Model solved by QCAC. The RUAC component collects the data from various resource s (e.g CPU, Disks) and also has a count on various completing request. This parameter helps in computing throughput. The average service time of a request at a resource can be computed as the ratio between the resource utilization and the system throughput. The QCAC generates the best possible configuration for AWS cloud considering the QoS inputs, the arriving requests and departing requests.

In the general approach for designing self managing / self organizing computer system [12], a system is subject to a work load. There are many factors of parameters and settings that may affect the performance of such systems. The set of parameters can be divided into uncontrolled parameters and controlled parameters. Uncontrolled parameters are that are not changed dynamically by controller. These parameters are those that have relatively little impact on performance or that require a system restart or reboot in order to take effect to take place. Controlled parameters are those whose settings are changed dynamically by the controller QCAC, by executing controller algorithm. The goal of this algorithm is to find optimized instance under present load, satisfying QoS values. A set of responses are generated as the result of service request load and setting of controlled and uncontrolled parameters. These responses can be divided into primary and secondary responses. The former are those whose values must be kept within desired range as specified by Service Layer Agreement (SLAs) or QoS goals

QCAC Algorithm:

QCAC algorithm for finding close to optimal configuration for instance allotment-

Step 1	If Service Requests in a particular interval is greater than > No of requests (n): go to Step 16 , else step 2
Step 2	Check WSDL for incoming service requests i.e., if P_1 :Go to step13 , if P_2 :Go to step10, if P_3:Go to step 7, if P_4 :Go to step 3,
Step 3	If P_4, Check for any other type of instance is being served presently, if yes, go to step 5, else Step 4
Step 4	Check QoS fitness satisfaction under present load, if satisfied choose P_{40} instance , else choose P_{33} instance
Step 5	Choose highest priority instance running at primary level, Check QoS fitness under present load is satisfied, If yes, choose the nearest matching priority instance, allot & continue , else go to Step 6
Step 6	Go to step 16
Step 7	If P_3, Check if any higher priority instance is running, If yes, go to step 5, else step 8
Step 8	Check QoS fitness satisfaction under present load is satisfied, if yes, choose P_{33},else step 9
Step 9	Choose the nearest matching instance i.e. P_{31} or P_{32}
Step 10	If P_2,check if any highest priority instance is running, if yes, go to step 5 ,else step 11
Step 11	Check QoS fitness satisfaction under present load is satisfied, if yes, choose P_{22} ,else step 12
Step 12	Choose the nearest matching instance i.e. P_{22} or P_{21}
Step 13	If P_1 , check if P_{11} type request is running, if yes, go to step15, else step 14
Step 14	Check for QoS fitness satisfaction under present load, if satisfied choose P_{12}, else Step 16
Step 15	Check for QoS fitness satisfaction under present load is satisfied, if yes continue with P_{11} ,else step 16
Step 16	Discard this request
	Choose Micro Instance (MI), P_{40}
	Choose Standard Small Instance (SSI), P_{33}
	Choose Standard Large Instance (SLI), P_{32}
	Choose Standard Extra Large Instance (SELI), P_{31}
	Choose High Memory Extra Large Instance (HMELI), P_{23}
	Choose High Memory Double Extra Large Instance (HMDELI), P_{22}
	Choose High Memory Quadruple Extra Large Instance (HMQELI), P_{21}
	Choose High CPU Medium Instance (HCMI),

	P_{12}
	Choose High CPU Extra large Instance (HCELI), P_{11}

VI. Experimental Results

Since EC2's servers are Linux based virtual machines running on top of the Zen Virtualization Engine [14]. A Linux machine having 1.7 GHz x 86 processor, 1.75 GB of RAM, 160 GB of local disk is used for experiments. The virtualization is implemented on this machine as Amazon web Services are implementing, using Xen based Virtualization Environment. This virtual environment is also used by Amazon. This environment uses Xen hypervisor, the Domain 0, and 9 VM guests. These nine VM guests implemented nine instances under consideration. The system consists of one CPU and one disk. The workload is being driven by another machine using proxy-sniffer (a workload generator), which can also be used for Amazon EC2. The service demands at the CPU and disks are 0.03 sec and 0.05 sec, respectively. The SLA and the respective weights are:

- $R \leq 1.3$ seconds and W_R=0.25,
- $X \geq 7$ requests/sec and w_x= 0.30, and
- $P_{rej} \leq 0.05$ and w_p

Table 1 details the instances with their respective weight. The QCAC algorithm is implemented periodically, for each interval. QCAC chooses the most optimum instance, satisfying the QoS, no of service request, workload conditions etc.

Table 1: Instances and their respective weights

Type	Notation	Sub division of instances
Micro Instance type	MI	Micro instance (MI), wt (.1), P_{40}
Standard Instance	SI	Standard Small instance (SSI), wt (.2), P_{33}
		Standard Large instance (SLI), wt (.3), P_{32}
		Standard Extra Large instance (SELI), wt (.4), P_{31}
High Memory Instance	HM	High Memory Extra Large instance (HMELI), wt (.5), P_{23}
		High Memory Double Extra Large instance (HMDELI), wt (.6), P_{22}
		High Memory Quadruple Extra Large instance (HMQELI), wt (.7), P_{21}
High CPU Instance	HC	High CPU Medium instance (HCMI), wt (.8), P_{12}
		High CPU Extra Large instance (HCELI), wt (.9), P_{11}

During experiments, the arrival rate of requests started from a low of 7 service requests per second and the load was increased up to a maximum of 23 service requests per second, during a period of 1 hr and 40 min. The controller interval is of 300 seconds. During any interval with peak average loads of 23 service requests per second, 6900 requests arrive. At the maximum load of 23 service request per second, the resource bottleneck reaches close to 100 %, after this load was not increased further otherwise the probability of rejection would be turning up too high.

After the arrival rate reaches to its maximum value of 23 service request per second, the disk utilization reaches to its max of 99% approx, where as for the uncontrolled case the peak disk utilization value is only of max 83%. When the controller is disabled not all the system resources are being utilized optimally. Due to the controller action the system resources are better utilized, furnishing required QoS. The controlled system adjusts itself to keep controlled even at higher loads, where as the uncontrolled system violates the SLA as soon as the service request reaches to value of 15 requests per second. Hence the controlled system satisfies the QoS also and gives proper computational facility while saving cost.

Fig. 1 explains the variation of average response time versus the arrival of number of requests per second.
It can be seen that as the number of service request per second reaches to its peak, the response time for controlled system moves to a value of 1.6 sec and even it violates the SLA for a very short duration. Whereas the response time for the uncontrolled system is lower than in comparison to the controlled system response time. The reason for this is that at high loads, even 22% of the service requests are rejected and are kept out of the queue. The controlled system adjusts itself to meet the SLA as close as possible, minimizing the probability of rejection at the same time.

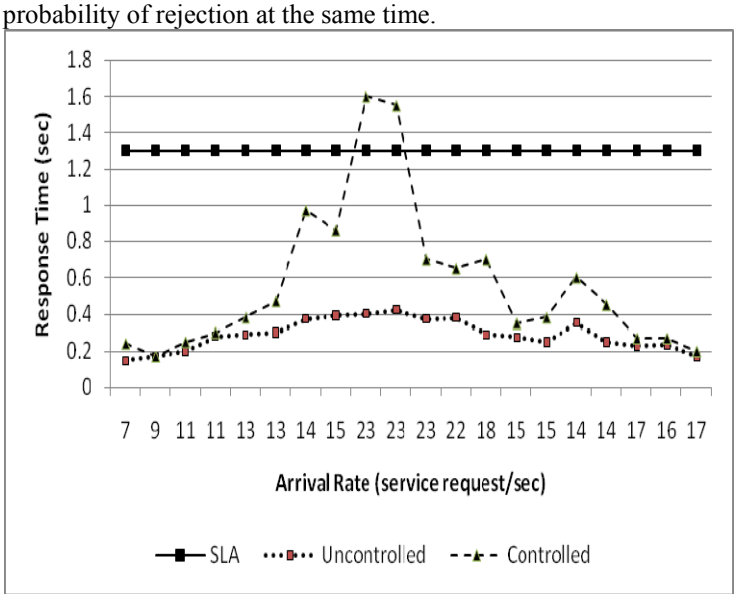

Figure 1: Response time under different environments.

This provides higher throughput (Fig.2). The throughput of the controlled system is better than the uncontrolled one, due the lower probability of rejection.

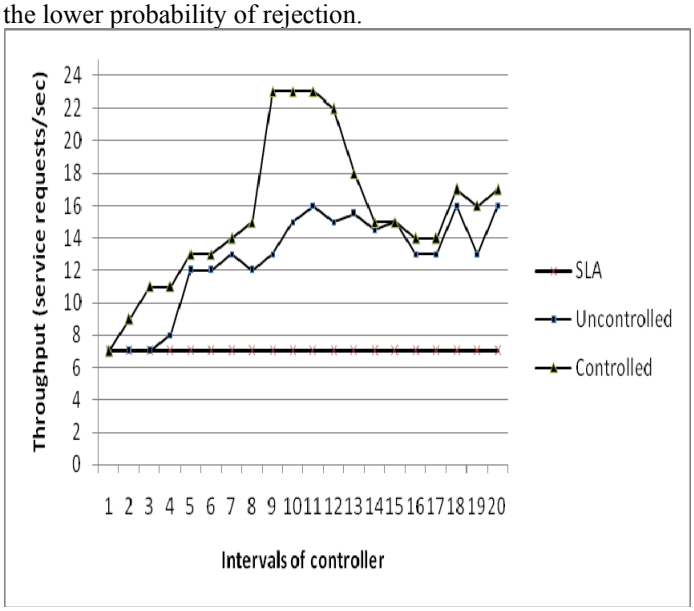

Figure 2: Throughput under controlled and Uncontrolled environment.

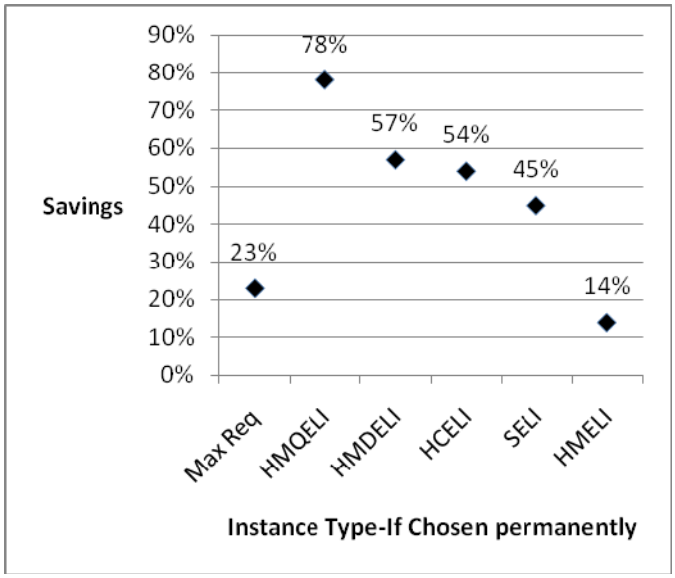

Figure 3:%Savings done using controller

Fig.3 gives a comparison of % Savings in the money to be given to providers, if different Instances are chosen dynamically. Having taken the highest computational capacity allotted permanently the savings are of 23%, but in this case the resource utilization is poorest. Where in other cases considering, allotting P21 permanently the saving is of 78% in controlled case, allotting P22 permanently the saving is of 57% in controlled case, allotting P11 permanently the saving is of 54% in controlled case, allotting P31 permanently the saving is of 45% in controlled case, allotting P21 permanently the saving is of 14% in controlled case. The experiments show that the controlled system performs better in term of saving the cost, giving optimum performance satisfying QoS, gives better throughput and response time.

VII. Conclusion

In this paper we present an approach using which dynamic allocation of instances can be done, which also satisfies QoS. This controller mechanism is particularly useful for consumer whose load varies continuously, and mostly their applications are not heavily loaded with requests. It also illustrates the concept of virtualization being used by large solution providers, for allotting various instances.

References

[1] www.amazonwebservice.com
[2] Menasce, D.A. and V.A.F. Almeida, Scaling for E- Business: technologies, models, performance and capacity planning, Prentice Hall, Upper Saddle River, NJ, 2002
[3] Kleinrock, L., Queuing systems, Vol. 1: Theory
[4] Kishore S. Trivedi, Probability & Statistics with Reliability, Queuing and Computer Science Applications, PHI
[5] S L Garfinkel, Technical Report TR-08-07: An Evaluation of Amazon's Grid Computing Services: EC2,S3 and SQS
[6] XenSource Inc. Xen. http://www.xensourse.com
[7] Amazon web services. Amazon simple storage service (amazon s3), May 2007. http://aws.amazon.com/s3
[8] K J Duda and D R Cheriton, " Borrowed- virtual- Time(bvt) scheduling: supporting latency sensitive threads in a general purpose scheduler" In proceedings of SOSP'99,Dec 1999.
[9] D A Menansce, M N Bennani, Honglei Ruan, "On the Use of Online Analytic Performance Models in Self Managing and Self-Organizing Computer Systems.
[10] D A Menansce, Almeida, V A F Dowdy, Performance by Design: Computer Capacity planning by Example, Prentice Hall, 2004
[11] XenSource Inc. Xen. http://www.xensource.com
[12] E. Walker ,"Benchmarking Amazon EC2for high performance scientific computing", USENIX- Oct 2008.
[13] Guohui Wang, T. S. Eugene Ng, The Impact of Virtualization on Network Performance of Amazon EC2 Data Centre, INFOCOM2010
[14] Y. Zhang, N.Duffield, V. Paxson, and S. Shenkar,"On The Constancy Of Internet Path Properties",in proceedings of IMW'01,Nov.2001
[15] D. Ersoz,M.S.Yousif and C.R. Das, "Characterizing Network Traffic In A Cluster Based, Multi-Tier Data Centre,"ICDCS'07
[16] B. Abraham, J. Leodolter, and J. Leododalter, "Statistical Methods of Forecasting,"John Wiley & Sons,1983
[17] D.A. Menasce, "Automatic QoS Control,"IEEE Internet Computing, Jan/Feb. 2003.
[18] V.J.Rayward-Smith, I.H.Osman, C.R.Reeves, eds ,Modern Heuristic Search Methods, John Wiley & Sons, Dec. 1996
[19] W.E. Walsh, G. Tesauro, J.O.Kephart, and R. Das, "Utility Functions in Autonomic Computing," Proc. IEEE International Conf. Autonomic Computing(ICAC'04), New York, 2004
[20] "Gogrid," http://www.gogrid.com/
[21] Simson L. Garfinkel, Technical Report TR-08-07:An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQ