ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Random Scaling of Internet Applications for Cloud Computing Services

E.Sravani
M.Tech Student,
Department of CSE,
Sree Rama institute of Technology and Science,
Kuppenakuntla, Penuballi, Khammam,TS India.

B.R.M.Reddy
Assistant Professor,
Department of CSE,
Sree Rama institute of Technology and Science,
Kuppenakuntla, Penuballi, Khammam,TS India.

ABSTRACT:

Many Internet applications can benefit from an automatic scaling property where their resource usage can be scaled up and down automatically by the cloud service provider. We present a system that provides automatic scaling for Internet applications in the cloud environment. We encapsulate each application instance inside a virtual machine (VM) and use virtualization technology to provide fault isolation. We model it as the Class Constrained Bin Packing (CCBP) problem where each server is a bin and each class represents an application. The class constraint reflects the practical limit on the number of applications a server can run simultaneously.

We develop an efficient semi-online color set algorithm that achieves good demand satisfaction ratio and saves energy by reducing the number of servers used when the load is low. Experiment results demonstrate that our system can improve the throughput by 180% over an open source implementation of Amazon EC2 and restore the normal QoS five times as fast during flash crowds. Large scale simulations demonstrate that our algorithm is extremely scalable: the decision time remains under 4 seconds for a system with 10,000 servers and 10,000 applications. This is an order of magnitude improvement over traditional application placement algorithms in enterprise environments.

Index Terms:

cloud computing, virtualization, auto scaling, CCBP, green computing.

INTRODUCTION:

ONE of the often cited benefits of cloud computing service is the resource elasticity: a business customer can scale up and down its resource usage as needed without upfront capital investment or long term commitment. The Amazon EC2 service [1], for example, allows users to buy as many virtual machine (VM) instances as they want and operate them much like physical hardware. However, the users still need to decide how much resources are necessary and for how long. We believe many Internet applications can benefit from an auto scaling property where their resource usage can be scaled up and down automatically by the cloud service provider. A user only needs to upload the application onto a single server in the cloud, and the cloud servicewill replicate the application onto more or fewer servers as its demand comes and goes. The users are charged only for what they actually use – the so-called "pay as you go" model.

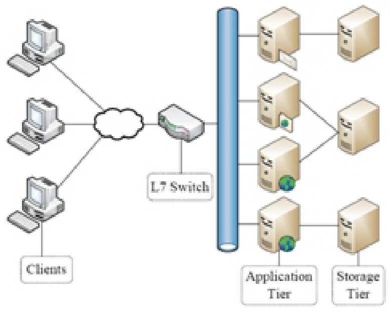

Fig. 1. Two-tiered architecture for Internet applications

Figure 1 shows the typical architecture of data center servers for Internet applications.

It consists of a load balancing switch, a set of application servers, and a set of backend storage servers. The front end switch is typically a Layer 7 switch [2] which parses application level information in Web requests and forwards them to the servers with the corresponding applications running.The switch sometimes runs in a redundant pair for fault tolerance. Each application can run on multiple server machines and the set of their running instances are often managed by some clustering software such as WebLogic [3]. Each server machine can host multiple applications. The applications store their state information in the backend storage servers. It is important that the applications themselves are stateless so that they can be replicated safely. The storage servers may also become overloaded, but the focus of this work is on the application tier. The Google AppEngine service, for example, requires that the applications be structured in such a two tier architecture and uses the BigTable as its scalable storage solution [4].

Existing System:

Even though the cloud computing model is sometimes advocated as providing infinite capacity on demand, the capacity of data centers in the real world is finite.The illusion of infinite capacity in the cloud is provided through statistical multiplexing. When a large number of applications experience their peak demand around the same time, the available resources in the cloud can become constrained and some of the demand may not be satisfied. We define the demand satisfaction ratio as the percentage of application demand that is satisfied successfully. The amount of computing capacity available to an application is limited by the placement of its running instances on the servers. The more instances an application has and the more powerful the underlying servers are, the higher the potential capacity for satisfying the application demand. On the other hand, when the demand of the applications is low, it is important to conserve energy by reducing the number of servers used.Various studies have found that the cost of electricity is a major portion of the operation cost of large data centers. At the same time, the average server utilization in many Internet data centers is very low: real world estimates range from 5% to 20% [5] [6]. Moreover, work [7] has found that the most effective way to conserve energy is to turn the whole server off. The application placement problem is essential to achieving a high demand satisfaction ratio without wasting energy.

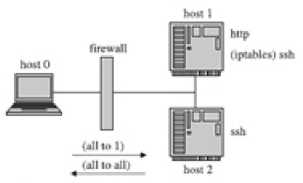

Fig. 1. An example network.

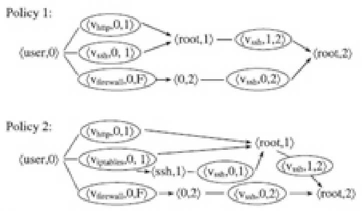

Fig. 2. Sequences of zero-day attacks.

Proposed System:

In this paper, we present a system that provides automatic scaling for Internet applications in the cloud environment. Our contributions include the following:
• We summarize the automatic scaling problem in the cloud environment, and model it as a modified Class Constrained Bin Packing (CCBP) problem where each server is a bin and each class represents
an application. We develop an innovative auto scaling algorithm to solve the problem and present a rigorous analysis on the quality of it with provable bounds. Compared to the existing Bin Packing solutions, we creatively support item departure which can effectively avoid the frequent placement changes 1 caused by repacking.
• We support green computing by adjusting the placement of application instances adaptively and putting idle machines into the standby mode. Experiments and simulations show that our algorithm is highly efficient and scalable which can achieve high demand satisfaction ratio, low placement change
frequency, short request response time, and good energy saving.

• We build a real cloud computing system which supports our auto scaling algorithm. We compare the performance of our system with an open source implementation of the Amazon EC2 auto scaling system in a testbed of 30 Dell PowerEdge blade servers. Experiments show that our system can restore the normal QoS five times as fast when a flash crowd happens.
• We use a fast restart technique based on virtual machine (VM) suspend and resume that reduces the application start up time dramatically for Internet
Services.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

SYSTEM ARCHITECTURE:

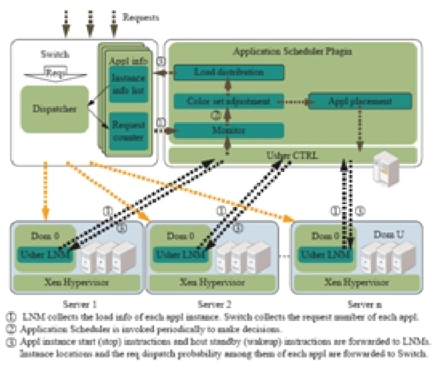
Requests
Switch
Req
Dispatcher
Appl info
Instance info list
Request counter
Application Scheduler Plugin
Load distribution
Color set adjustment
Appl placement
Monitor
Usher CTRL
Dom 0
Usher LNM
Dom U
Xen Hypervisor
Server 1
Server 2
Server n
① LNM collects the load info of each appl instance. Switch collects the request number of each appl.
② Application Scheduler is invoked periodically to make decisions.
③ Appl instance start (stop) instructions and host standby (wakeup) instructions are forwarded to LNMs. Instance locations and the req dispatch probability among them of each appl are forwarded to Switch.

Fig. 2. Overview of the System Architecture

The architecture of our system is shown in figure 2. We encapsulate each application instance inside a virtual machine (VM). The use of VMs is necessary to provide isolation among untrusted users. Both Amazon EC2 and Microsoft Azure use VMs in their cloud computing offering. Each server in the system runs the Xen hypervisor which supports a privileged domain 0 and one or more domain U [8]. Each domain U encapsulates an application instance, which is connected to a shared network storage (i.e., the storage tier). The multiplexing of VMs to PMs (Physical Machines) is managed using the Usher framework [9]. (We use the terms "server", "PM", and "node" interchangeably in this paper.) The main logic of our system is implemented as a set of plug-ins to Usher.

Each node runs a Usher local node manager (LNM) on domain 0 which keeps track of the This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. set of applications running on that node and the resource usage of each application. A L7 switch is in charge of forwarding requests and responses. The schedule procedure of our system can be described as follows:

• The LNM at each node and the L7 switch collect the application placement, the resource usage of each instance, and the total request number of each application periodically. Then the information is forwarded to the Usher central controller (Usher CTRL) where our "Application Scheduler" runs.

• The Application Scheduler is invoked periodically to make the following decisions:

– application placement: for each application, we need to decide the set of servers its instances run on.

– load distribution: for each application, we need to predict its future resource demands based on the request rate and past statistics, and then decide how to allocate its load among the set of running instances. The load of an Internet application is largely driven by the rate of user requests. We profile the application to estimate the average load incurred by each request. Then we analyze the queues of pending requests in L7 switch to predict the load on the servers.

• The decisions are forwarded to the LNM and the L7 switch for execution. The list of action items for each node includes:

– standby or wake up instructions

– application starts and stops

– the allocation of local resource among the applications

The LNM at the node adjusts the local resource allocation of the VMs encapsulating the applications.

Xen can change the CPU allocation among the VMs by adjusting their weights in its CPU scheduler. Memory allocation among the VMs can be adjusted using the ballooning technique. After that the Scheduler notifies the L7 switch of the new configuration including:

– the list of applications

– for each application, the location of its running instances and the probability of request distribution among them

The L7 switch then starts processing Web requests according to the new configuration.

The decision interval of the Scheduler depends on how responsive we want to be to application demand change. Frequent placement changes are disruptive to application performance and should be avoided.

RELATED WORK:

The traditional bin packing problem has been extensively studied in the literature (see the survey in[19]). The vector bin packing problem considers multidimensional constraints when packing items into a minimum number of bins [20]. One may think we can consider the CPU demand and the memory requirement of an Internet application as individual elements in thevector and use vector bin packing to solve our problem. Unfortunately, the memory requirement of Internet applications has to be satisfied as a whole: a major portion of the memory is consumed anyway even when the application receives little load. This is especially true for Java applications whose memory usage may depend on the past load due to garbage collection. Hence, we cannot divide the memory requirement and satisfy it in a piecemeal manner across the servers.

None of theexisting bin packing problems can be applied in our environment. The Class Constrained Multiple Knapsack problem (CCMK) aims to maximize the total number of packed items under the restriction that each knapsack has a limited capacity and a bound on the number of different types of items it can hold [21], [22]. Unlike CCBP, it does not attempt to minimize the number of knapsacks used. Hence, unlike our algorithm, it does not support green computing when the system load is low.A number of approximation algorithms have been developed for CCBP. Most of them are offline algorithms which do not support item departure. The rest are strict online algorithms which do not allow movements of already packed items. In the case of item departure, thedeparted item is removed but the rest of the items in the bins are not repacked. When a color set becomes unfilled due to application leaves, those algorithms do not maintain the property that there is at most one unfilled color set in the system. This can degrade the performance severely because each color set is packed independently. It has been shown that the existing color set algorithms perform poorly in the face of frequent item departure [13]. They cannot be applied in a cloud computing environment where the application demands change dynamically.

Resource provisioning for Web server farms has been investigated in [23], [24], [25], [26]. Some allocate resourcesin the granularity of whole servers which can lead to inefficient resource usage. Some do not consider the practical limit on the number of applications a server can run simultaneously [25]. Bhuvan et al. support shared hosting, but manage each application instance independently [23]. They do not provide the auto-scaling property. Mohit et al. group applications into service classes which are then mapped onto server clusters [24]. However, they do not attempt to minimize the placement changes when application demands vary and is mostlyfor offline use. Zhang et al. organize a set of shared clusters into a network and study resource allocation across shared clusters [26], which is not the focus of this paper.Process migration has been studied in various contexts, e.g., [27]. Unlike virtualization technology, it does not capture the execution environment of the running processes. Nor does it support the auto scaling of the processes based on the observed demand. Application placement in enterprise environments has been studied in [28], [29], [30], [31]. They run multiple applications on the same set of servers directly without using VMs or Sandbox. Their approach is suitable when the applications are trustworthy (e.g., enterprise applications).

It is not suitable for a cloud environment where applications come from untrusted users. Unlike ours, their decision algorithm has no concern on green computing and is based on a set of heuristics with no provable bounds or optimality. Our algorithm can scale to an order of magnitude more servers than those in [28], [29] because the complexity of our algorithm is much lower.Like our system, the Google AppEngine service provides automatic scaling for Web applications. The users are charged by the CPU cycles consumed, not by the number of application instances. Its internal algorithm used is not disclosed. Our algorithm potentially can be used to implement such a service.

The applications in AppEngine must run inside a sandbox with severe restrictions on what they can do. At the time of this writing, it supports mostly applications written in Java and Python 5 or Google's own Go programming language.This makes it difficult to port legacy applications onto their platform. In contrast, porting an existing application onto our VM platform is much easier. Itgives the users great flexibility in choosing their favorite programming languages, operating systems, libraries,etc.. There are also some cloud vendors providing autoscaling solutions for cloud users (see the survey in [32]). Users are allowed to define a set of rules to control the scaling actions. However, the rules and the load balancing strategies they used are very simple. Just like the Scalr in Amazon EC2 [17], they perform the scaling actions simply when some conditions are met and balance the load evenly across all instances. Since they do not take the state of the whole system into consideration, they cannot reach a globally optimal decision.

CONCLUSIONS:

We presented the design and implementation of a system that can scale up and down the number of application instances automatically based on demand.We developed a color set algorithm to decide the application placement and the load distribution. Our system achieves high satisfaction ratio of application demand even when the load is very high. It saves energy by reducing the number of running instances when the load is low.There are several directions for future work. Some cloud service providers may provide multiple levels of services to their customers. When the resources become tight, they may want to give their premium customers a higher demand satisfaction ratio than other customers.

In the future, we plan to extend our system to support differentiated services but also consider fairness when allocating the resources across the applications. We mentionedin the paper that we can divide multiple generations of hardware in a data center into "equivalence classes" and run our algorithm within each class. Our future work is to develop an efficient algorithm to distribute incoming requests among the set of equivalence classes and to balance the load across those server clusters adaptively. As analyzed in the paper, CCBP works well when the aggregate load of applications in a color set is high. Another direction for future work is to extend the algorithm to pack applications with complementary bottleneck resources together, e.g., to co-locate a CPU intensive application with a memory intensive one so that different dimensions of server resources can be adequately utilized.

REFERENCES:

[1] "Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/."
[2] A. Cohen, S. Rangarajan, and H. Slye, "On the performance of tcpsplicing for url-aware redirection," in Proc. of the 2nd conference onUSENIX Symposium on Internet Technologies and Systems, 1999.
[3] "WebLogic, http://www.oracle.com/appserver/weblogic/weblogicsuite.html."[4] "Google App Engine, http://code.google.com/appengine/."
[5] M. Armbrust et al., "Above the clouds: A berkeley view ofcloud computing," EECS Department, University of California,Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.
[6] L. Siegele, "Let it rise: A special report on corporate IT," in TheEconomist, Oct. 2008.
[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, andR. P. Doyle, "Managing energy and server resources in hostingcenters," in Proc. of the ACM Symposium on Operating SystemPrinciples (SOSP'01), Oct. 2001.
[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho,R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the artof virtualization," in Proc. of the ACM Symposium on OperatingSystems Principles (SOSP'03), Oct. 2003.
[9] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: Anextensible framework for managing clusters of virtual machines,"in Proc. of the Large Installation System Administration Conference(LISA'07), Nov. 2007.
[10] J. Zhu, Z. Jiang, Z. Xiao, and X. Li, "Optimizing the performanceof virtual machine synchronization for fault tolerance," IEEETransactions on Computers, Dec. 2011.
[11] "RUBiS, http://rubis.ow2.org/."
[12] "Linux Documentation, http://www.kernel.org/doc/documentation/power/states.txt."
[13] H. Shachnai and T. Tamir, "Tight bounds for online classconstrainedpacking," Theor. Comput. Sci., vol. 321, no. 1, pp. 103–123, 2004.
[14] L. Epstein, C. Imreh, and A. Levin, "Class constrained bin packingrevisited," Theor. Comput. Sci., vol. 411, no. 34-36, pp. 3073–3089,2010.
[15] E. C. Xavier and F. K. Miyazawa, "The class constrained binpacking problem with applications to video-on-demand," Theor.Comput. Sci., vol. 393, no. 1-3, pp. 240–259, 2008.
[16] M. R. Garey and D. S. Johnson, "A 71/60 theorem for binpacking," Journal of Complexity, vol. 1, 1985.
[17] "Scalr: the auto scaling open source Amazon EC2 effort,https://www.scalr.net/."
[18] D. Magenheimer, "Transcendent memory: A new approach tomanaging RAM in a virtualized environment," in Linux Symposium,2009.
[19] G. Galambos and G. J. Woeginger, "On-line bin packing-a restrictedsurvey," Physica Verlag, vol. 42, no. 1, 1995.
[20] C. Chekuri and S. Khanna, "On multidimensional packing problems,"SIAM J. Comput. Issue 4, vol. 33, 2004.
[21] H. Shachnai and T. Tamir, "Noah's bagels-some combinatorial aspects," in Proc. 1st Int. Conf. on Fun with Algorithms, 1998.
[22] ——, "On two class-constrained versions of the multiple knapsackproblem," Algorithmica, vol. 29, no. 3, pp. 442–467, 2001.
[23] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbookingand application profiling in shared hosting platforms," SIGOPSOper. Syst. Rev., vol. 36, no. SI, pp. 239–254, 2002.
[24] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: amechanism for resource management in cluster-based networkservers," SIGMETRICS Perform. Eval. Rev., vol. 28, no. 1, pp. 90–101, 2000.
[25] J. L. Wolf and P. S. Yu, "On balancing the load in a clustered webfarm," ACM Trans. Internet Technol., vol. 1, no. 2, pp. 231–261, 2001.
[26] C. Zhang, V. Lesser, and P. Shenoy, "A Multi-Agent LearningApproach to Online Distributed Resource Allocation," in Proc. ofthe International Joint Conference on Artificial Intelligence (IJCAI'09),2009.
[27] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The design andimplementation of zap:

a system for migrating computing environments,"SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 361–376, 2002.
[28] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder,M. Sviridenko, and A. Tantawi, "Dynamic placement for clusteredweb applications," in Proc. of the International World Wide WebConference (WWW'06), May 2006.
[29] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalableapplication placement controller for enterprise data centers," inProc. of the International World Wide Web Conference (WWW'07),May 2007.
[30] C. Adam and R. Stadler, "Service middleware for self-managinglarge-scale systems," IEEE Transactions on Network and ServiceManagement, vol. 4, no. 3, pp. 50–64, 2007.
[31] J. Famaey, W. D. Cock, T. Wauters, F. D. Turck, B. Dhoedt, andP. Demeester, "A latency-aware algorithm for dynamic serviceplacement in large-scale overlays," in Proc. of the IFIP/IEEE internationalconference on Symposium on Integrated Network Management(IM'09), 2009.
[32] E. Caron, L. Rodero-Merino, F. Desprez, and A. Muresan, "Autoscaling,load balancing and monitoring in commercial and opensourceclouds," INRIA, Rapport de recherche RR-7857, Feb. 2012.

Author's:

E.Sravani is a student of Sree Rama Institute of Technology & Science, Kuppenakuntla,Penuballi, Khammam, TS,India.Presently she is Pursuing her M.Tech (CSE) from this collegeHer area of interests includes Information Security, Cloud Computing, Data Communication & Networks.

Mr. B.R.M.Redy is an efficient teacher, received M.Tech from JNTU Hyderabad is working as an Assistant Professor in Department of C.S.E, Sree Rama Institute of Technology & Science, Kuppenakuntla, Penuballi, Khammam, AP,India. He has published many papers in both National & International Journals. His area of Interest includes Data Communications & Networks, Information security, Database Management Systems, Computer rganization, C Programming and other advances in Computer Applications