International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-4, July 2015
ISSN: 2395-3470
www.ijseas.com

AUTOMATIC SCALING OF INTERNET APPLICATIONS FOR CLOUD COMPUTING SERVICES

Kotta Pratap Reddy[1], Mr.Y.Durga Prasad[2], Dr.Y.Venkateswarulu[3]

[1]MtechStudent ,CSE, Giet Engineering College, Rajahmundry, A,P, India
[2]Asst. Professor, Dept of CSE, Giet Engineering College, Rajahmundry, A,P, India
[3]Professor and HOD, Dept of CSE, Giet Engineering College, Rajahmundry, A,P, India

ABSTRACT

An automatic scaling property utilized by many Internet applications in the cloud service provider and get benefit from where their resource usage can be scaled up and down automatically.The present paper proposes a system that provides automatic scaling for Internet applications in the cloud environment. The present method encapsulate each application instance inside a virtual machine (VM) and use virtualization technology to provide fault isolation and this method is called 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. The present paper develops 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 the proposed 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.

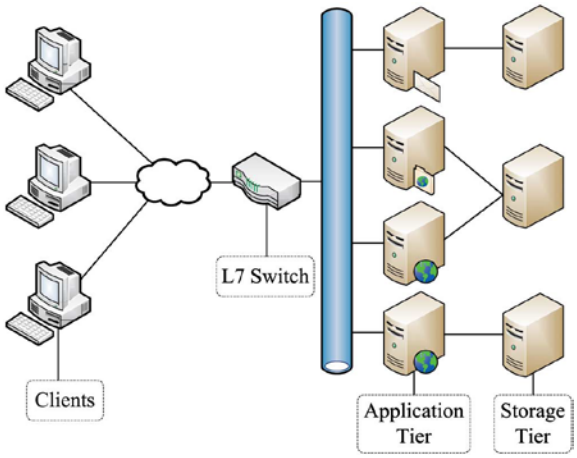
Index Terms—Cloud computing, virtualization, auto scaling, CCBP, green computing

INTRODUCTION

One of the most usefulbenefits of cloud computing service is the resource flexibility: 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. A user only needs to upload the application onto a single server in the cloud, and the cloud service will replicate the application onto more or fewer servers as its demand comes and goes.

Fig. 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. 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. 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].



Fiure 1: Internet applications two-tiered architecture

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. The present method defines 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. 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].

In this paper, we present a system that provides automaticscaling for Internet applications in the cloud environment.The present approach includes the following.The automatic scaling problem in thecloud environment, and model it as a modified ClassConstrained Bin Packing (CCBP) problem where eachserver is a bin and each class represents an application.The present study develop an innovative auto scaling algorithm to solvethe problem and present a rigorous analysis on the qualityof it with provable bounds. Experiments and simulationsshow that our algorithm is highly efficient andscalable which can achieve high demand satisfactionratio, low placement change frequency, short requestresponse time, and good energy saving.

The present paper builds a real cloud computing system which supportsour auto scaling algorithm. the present method compare the performanceof our proposed system with an open source implementation of theAmazon EC2 auto scaling system in a testbed of 30 DellPowerEdge blade servers. Experiments show that the proposedsystem can restore the normal QoS five times as fast whena flash crowd happens.

The rest of the paper is organized as follows. Section 2presents the architecture of the system and formulatesthe auto scaling problem. Section 3 describes the details of our proposed algorithm. Experiment results are presentedin Sections 4. Section 7 gives the conclusions of the paper.

SYSTEM ARCHITECTURE

The architecture of our system is shown in Fig. 2. The proposed method encapsulates each application instance inside a virtualmachine (VM). The use of VMs is necessary to provideisolation among untrusted users. Each server in the system

runs the Xen hypervisor whichsupports a privileged domain 0 and one or more domainU [7]. Each domain U encapsulates an application instance,which is connected to shared network storage (i.e., thestorage tier). The multiplexing of VMs to PMs (PhysicalMachines) is managed using the Usher framework [8]. The main logic of our system is implementedas a set of plug-ins to Usher. Each node runs an Usher localnode manager (LNM) on domain 0 which keeps track of theset of applications running on that node and the resourceusage of each application.

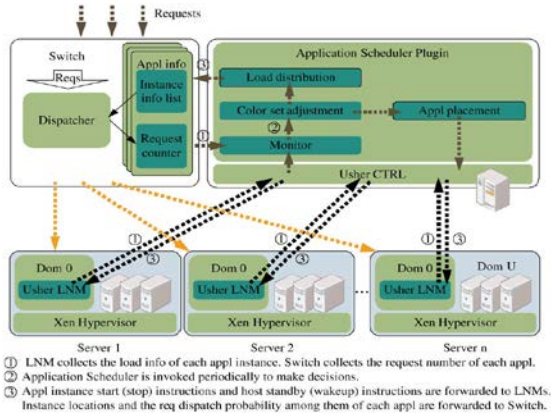


Figure 2: the system architecture.

The schedule procedure of our system can be described asfollows.

1. The LNM at each node and the L7 switch collectthe application placement, the resource usage of eachinstance, and the total request number of each applicationperiodically.
2. The Application Scheduler is invoked periodically tomake the following decisions:

   — Application placement: for each application decide the set of servers its instances run on.

   — load distribution: for each application, predict its future resource demands based on therequest rate and past statistics, and then decide how to allocate its load among the set of runninginstances. The load of an Internet application islargely driven by the rate of user requests.

3. The decisions are forwarded to the LNM and the L7switch for execution. The list of action items for eachnode includes:

   — standby or wake up instructions

   — application starts and stops

   — the allocation of local resource among the applications

After that the Scheduler notifies the L7 switchof the new configuration including:

— the list of applications

— for each application, the location of its runninginstances and the probability of request distributionamong them

The L7 switch then starts processing Web requests accordingto the new configuration.It may seem from the discussion above that the UsherCTRL is a central point of failure.

From above architecture notice that complicated applications can take along time (several minutes or much longer) to start and finishall the initializations.The present method takes the advantage of this featureto bypass the application start process by suspending a fullystarted and initialized application instance to the disk

PROPOSED METHOD

Automatic scaling: The auto scaling problem is defined as follows: Suppose, have a server set *S* on which need torun a set of applications (A). The CPU capacity ofserver s (s € *S*) is C_s, the maximum number of applicationinstances which can run on server simultaneously accordingto memory factor is M_s, and the CPU demand of application(a € *A*) isC_a .

To simplify the problem described above, the present method assumption that the servers are homogeneous with uniform capacity. Then the auto scaling problem is similar to the Class Constrained Bin Packing (CCBP) problem when label each application as a class and treat the CPU demands of all classes as the items which need to be packed into bins. The only difference is that the CCBP problem does not have the "Minimize the placement change frequency" goal. Therefore, in order to solve our problem, the present method modified the CCBP model to support the "Minimize the placement change frequency" goal and provide a new enhanced semi online approximation algorithm to solve it .the proposed algorithm is given below.

Proposed Algorithm: the proposed algorithm belongs to the family of color set algorithms [13], but with significant modification to adapt to proposed problem. The proposed methods labels each class of items with a color and organize them into color sets as they arrive in the input sequence. The number of distinct colors in a color set is at most c (i.e., the maximum number of distinct classes in a bin).This ensures that items in a color set can always be packed intothe same bin without violating the class constraint. Thepacking is still subject to the capacity constraint of the bin.All color sets contain exactly c colors except the last one whichmay contain fewer colors.

Items from different color sets are packed independently.Agreedy algorithm is used to pack items within each color set:the items are packed into the current bin until the capacity isreached. Then the next bin is opened for packing. Thus eachcolor set has at most one unfilled (i.e., non-full) bin. Note that a full bin may contain fewer than c colors. When a new itemfrom a specific color set arrives, it is packed into the correspondingunfilled bin. If all bins of that color set are full, then anew bin is opened to accommodate the item.

Application Load Increase

The load increase of an application is modeled as the arrival of items with the corresponding color. A naive algorithm is to always pack the item into the unfilled bin if there is one. If the unfilled bin does not contain that color already, then a new color is added into the bin. This corresponds to the start of a new application instance which is an expensive operation. Instead, the proposed algorithm attempts to make room for the newitem in a currently full bin by shifting some of its items into theunfilled bin. Let be the color of the new item and be any ofthe existing colors in the unfilled bin. The proposed method search for a binwhich contains items of both colors. Then the proposed method moves an item of color from bin to the unfilled bin. Thismakes room for an item in bin where we pack the new item.

The searching for bin process is

$binb_1$ contains colors c_1 and c_3

$binb_2$ contains colors c_2 and c_3

If proposed method can find such two bins, the following procedureis follows:

- move an item of color c_2 from bin b_2 to the unfilled bin
- move an item of color c_3 from bin b_1 to bin b_2
- pack the item in bin b_1

This process is illustrated in Fig. 3

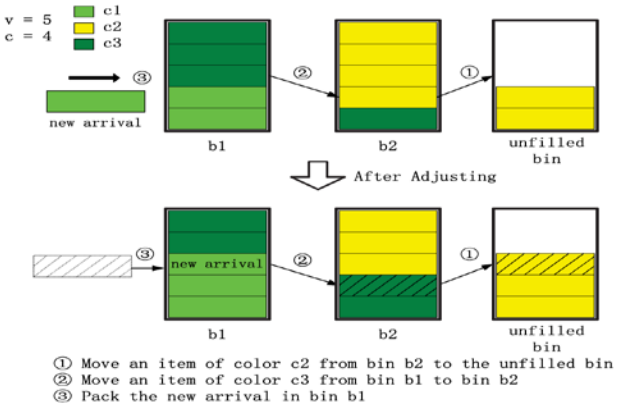


Figure 3: Schematic diagram ofArrival a new item

APPLICATION LOAD DECREASE

The load decrease of an application is modeled as the departure of previously packed items. Note that the departure event here is associated with a specific color, not with a specific item. The algorithm has the freedom to choose which item of thatcolor to remove.

The proposed departure algorithmworks as follows. If the color set does not have an unfilled bin,we can remove any item of that color and the resulting binbecomes the unfilled bin. Otherwise, if the unfilled bin containsthe departing color, a corresponding item there can beremoved directly. In all other cases, we need to remove anitem from a currently full bin and then fill the hole with anitem moved in from somewhere else. Let c_1 be the departingcolor and be any of the colors in the unfilled bin. The proposed method need tofind a bin which contains items of both colors. Let be such abin. We remove the departing item from bin and then movein an item of color from the unfilled bin. The procedure issimilar to the previous case for application load increase. Fig. 4illustrates this process for a chain with three colors.

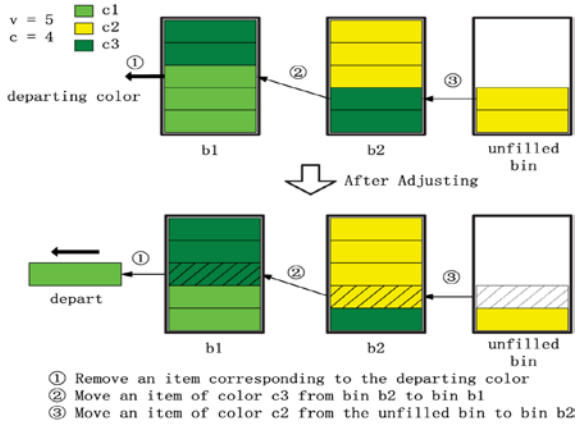


Figure 4: Schematic diagram of depicturing an existing item

Analysis of the Approximation Ratio

The quality of a polynomial time algorithm A is measured by its approximation ratio R(A) to the optimal algorithm OPT:

$$R(A) = \lim_{n\to\infty} \sup_{OPT(\sigma)=n} \frac{A(\sigma)}{OPT(\sigma)}$$

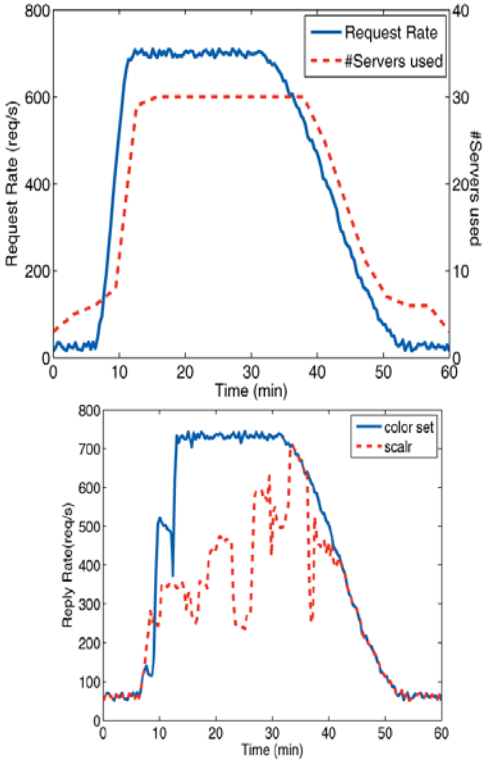
Where □ is the list and A(□) and OPT(bins used under the algorithm and the optimal algorithm, respectively [9, 10].

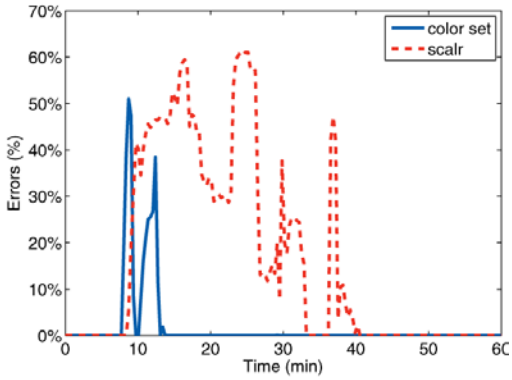
EXPERIMENTS

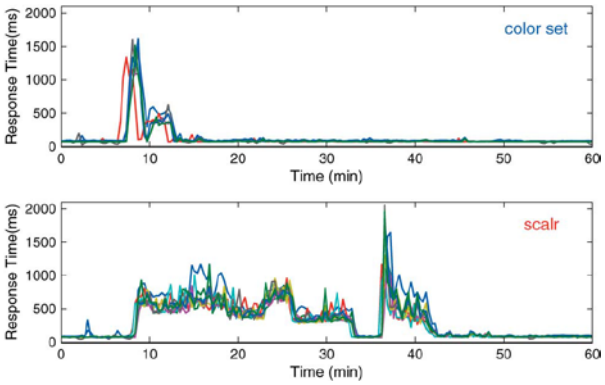
We in experiments.

To evaluate the effectiveness of our system, the Web applications used in the experiments are Apache servers serving CPU intensive PHP scripts. Each applicationinstance is encapsulated in a

separate VM. The servers areconnected over a Gigabit Ethernet. The client machines runhttperf to invoke the PHP scripts on the Apache servers. Thisallows us to subject the applications to different degrees ofCPU load by adjusting the client request rates. The proposed method considers aserver as “full” when its capacity reaches 80%. This leavessome room for additional load increase. To save time on the experiments, the present approach configures the Application Scheduler with anaggressive two minutes interval between invocations. Thisallows us to complete the experiments in a timely manner.







The auto scaling capability of our algorithm with nine applications and 30 Dell PowerEdge servers with Intel E5620 CPU and 24 GB of RAM. The servers run Xen-4.0 and Linux 2.6.18. The results are shown in Figs.5, 6, 7 and 8. To increase the load of one application dramatically to emulate a “flash crowd” event while keeping the load of the other applications steady.

Fig. 5 shows the request rate of theflash crowd application and the number of active servers (i.e.,APMs) used by all applications over the course of the experiment.Initially, the load in the system is low and only a smallnumber of servers are used. When the flash crowd happens, the present approached algorithm detects the skyrocketing request rate quicklyand scales up the server resources decisively. The above figures show that it uses up all 30 servers during the peak demand. Then the present method

reduces the request rate of the application graduallyto emulate that the flash crowd is over. The algorithm scalesdown the server resources accordingly to conserve energy.

CONCLUSIONS:

In this paper proposed the design and implementation of a system that can scale up and down the number of application instances automatically based on demand. In this paper developed a color set algorithm to decide the application placement and the load distribution. The proposed 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.

REFERENCES:

[1] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/. Accessed on May 10, 2012.

[2] A. Cohen, S. Rangarajan, and H. Slye, "On the performance of tcpsplicing for url-aware redirection," in Proc. 2nd Conf. USENIX Symp.Internet Technol. Syst., 1999, p. 11.

[3] Oracle WebLogic Suite. http://www.oracle.com/us/products/middleware/ cloud-app-foundation/weblogic/overview/index.html. Accessedon May 10, 2012.

[4] Google App Engine. http://code.google.com/appengine/. Accessedon May 10, 2012.

[5] M. Armbrust et al., "Above the clouds: A Berkeley view of cloudcomputing," EECS Depart., Univ. California, Berkeley, CA, Tech.Rep. UCB/EECS-2009–28, Feb. 2009.

[6] L. Siegele, "Let it rise: A special report on corporate IT," in TheEconomist, London, U.K.: London Economist Newspaper, Oct. 2008,vol. 389, pp. 3–16.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho,R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art ofvirtualization," in Proc. ACM Symp. Oper. Syst. Princ. (SOSP'03),Oct. 2003, pp. 164–177.

[8] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: Anextensible framework for managing clusters of virtual machines," inProc. Large Install. Syst. Admin. Conf. (LISA'07), Nov. 2007, pp. 1–15

[9] M. R. Garey and D. S. Johnson, "A 71/60 theorem for bin packing,"J. Complexity, vol. 1, pp. 65–106, 1985.

[10] Scalr: The Auto Scaling Open Source Amazon EC2 Effort. https://www.

scalr.net/. Accessed on May 10, 2012.