Automatic Scaling In Cloud Environment

[1]Vipin T K, [2]Sindhu S
[1]Mtech Student, [2]Professor
Computer Science and Engineering,
NSS College of Engineering, Palakkad, Kerala, India

***Abstract* - An energy-efficient load balancing technique can be used to improve the performance of Cloud Computing by balancing the workload across all the nodes in the cloud with maximum resource utilization and reducing energy consumption. This paper presents a system that provides automatic scaling for Internet applications in the cloud environment. A two level hierarchical load balancing model for cloud is proposed with a Global Centralized Scheduling Center (GCSC) at higher level and the Local Centralized Scheduling Center (LCSC) at lower level. The Global Centralized Scheduling Center uses the agent based adaptive balancing. The revenue of cloud provider can be increased by allocating the task to the appropriate server which executes the task with minimum cost and without violating the QOS constraints. It distributes the system workload based on the processing elements capacity which leads to minimize the overall job mean response time and maximize the system utilization and throughput. The Local Centralized Scheduling Center is modelled as the Class Constrained Bin Packing (CCBP) problem where each server is a bin and each class represents an application. It achieves good demand satisfaction ratio and saves energy by reducing the number of servers used when the load is low.**

***Keywords* - Cloud Computing, Automatic Scaling, Demand Satisfaction Ratio, Energy Conservation, Load balancing**

I. INTRODUCTION

Cloud computing is a general term for the delivery of hosted services over the Internet. Cloud Computing is a service model where resources (computing power, software, platform, storage, network) are offered in the form of a service on –demand to the consumers. It is a form of utility computing where users pay only for what they use. It has moved computing and data away from desktops into large datacenters. Cloud computing enables organizations to consume compute as an utility – like electricity or a telephone service – instead of building and maintaining computing infrastructures.

There are three cloud delivery models, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

Platform as a Service model is the delivery of a computing platform over the web where client can make and install their own particular application as their need. Configuration of computing platform and server is controlled by seller or cloud supplier. Example is Google App Engine.

Software as a Service Cloud Computing model is the delivery of a SAAS service where client don't have to control establishment and setup of any hardware or software. All the establishment and arrangement of administrations are controlled by seller or cloud supplier. Example is Email cloud.

Infrastructure of servers, programming, and network equipment is given as an on-demand service by the cloud supplier in Infrastructure as a service. Example is Amazon EC2.

A cloud is classified as private, public or hybrid. In a private cloud the computing infrastructure is dedicated to a particular organization and not shared with other organizations. Private clouds are more expensive and more secure when compared to public clouds.

In Public cloud the computing infrastructure is hosted by the cloud vendor at the vendor's premises. The customer has no visibility and control over where the computing infrastructure is hosted. The computing infrastructure is shared between any organizations.

Hybrid cloud Organizations may host critical applications on private clouds and applications with relatively less security concerns on the public cloud. The usage of both private and public clouds together is called hybrid cloud.

A. Cloud Architecture

The cloud framework is represented in the Fig. 1.1. Cloud utilizes hypervisor also called virtual machine manager (VMM), is one of numerous hardware virtualization strategies allowing various operating systems, to run simultaneously on a host PC.

The hypervisor presents to the guest operating systems as a virtual operating platform and deals with the execution of the guest operating systems. Different instances of the operating systems may share the virtualized hardware resources. Hypervisors are ordinarily used in server hardware, to run guest operating systems, which themselves act as servers.

The Cloud Computing is an infrastructure where resources will be dispersed through the network and services pooled together. Cloud is getting to be more popular because of its flexible nature, were the clients can access and release resources on-demand and pay just for the resources they need (pay-as-you-go model).

Cloud Computing concentrates on maximizing the effectiveness of the shared resources. Cloud resources are normally shared by different clients as well as dynamically reallocated per demand. Those resources are ordinarily as Virtual Machines (VM). One

of the regularly cited benefits of Cloud Computing services is the resource flexibility: a business client can scale up and down its resource use as required without upfront capital investment or long term commitment.

Numerous Internet applications can take advantage from an auto scaling property where their resource utilization can be scaled up and down naturally by the cloud service supplier. The Amazon EC2 service, for instance, permits clients to purchase as numerous virtual machines (VM) instances as they need and work them much like physical hardware. The clients still need to choose the amount of resources are vital and for to what extent. A client just needs to transfer the application onto a single server in the cloud, and the cloud service will reproduce the application onto more or less servers as its request goes comes and goes. The clients are charged just for what they really utilize the so-called "pay as you go" model.

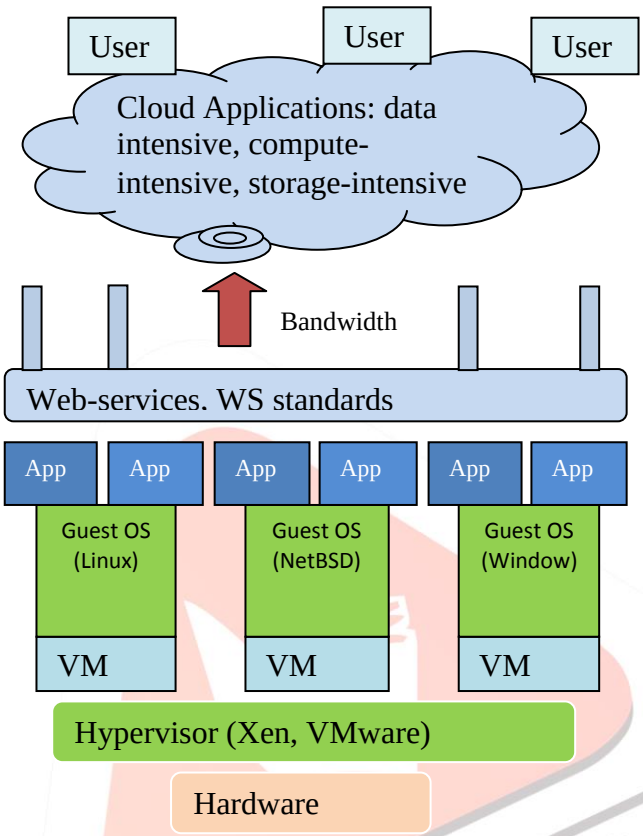

Fig. 1.1 cloud system

Despite the fact that the Cloud Computing model is some of the time advocated as giving infinite capacity on demand, the capacity of server farms in this present reality is limited. The illusion of infinite capacity in the cloud is provided through statistical multiplexing. At the point when an extensive number of applications experience their peak demand around the same time, the accessible resources in the cloud can get to be constrained and a percentage of the demand may not be fulfilled. The demand fulfillment proportion is characterized as the rate of application request that is fulfilled effectively. The amount of computing capacity accessible to an application is constrained by the number of its running instances on the servers. The more instances an application has and the all the more effective the servers are, the higher the potential limit for fulfilling the application request. Then again, when the demand of the application is low, it is important to save energy by reducing the number of servers utilized. The best approach to conserve energy is to turn the entire server off. The application placement problem is to accomplishing a high demand satisfaction ratio without wasting energy.

The system provides automatic scaling to Internet applications in the cloud environment. The automatic scaling problem is summarized in the cloud environment, and modeled it as a Class Constrained Bin Packing (CCBP) problem where every server is a bin and each one class represents an application. Contrasted with the current Bin Packing arrangements, it creatively supports item departure which can effectively avoid the frequent placement changes caused by repacking. It supports green computing by changing the position of application instances adaptively and putting idle machines into the standby mode. It uses a quick restart method focused around virtual machine (VM) suspend and resume that diminishes the application start up time significantly for Internet services.

II. RELATED WORKS

There are additionally some cloud vendors giving auto-scaling for cloud clients. Clients are permitted to characterize a set of standards to control the scaling activities. The guidelines and the load balancing procedures they utilized are simple.

The Automatic Scaling of Internet Applications for Cloud Computing Services [1] gives automatic scaling to Web applications. It can automatically scale the resources up and down. But it can only be applied on homogeneous servers with uniform capacity. Also the datacenters are vulnerable to overloading of the thermal system during the high service load. High temperature leads to a number of problems, such as reduced system reliability and availability, as well as decreased lifetime of

devices.

The Scalr in Amazon EC2 [2], Scalr is an open-sourced framework for managing the massive serving power of Amazon's Elastic Computing Cloud (EC2) service. Scalr can increase / decrease capacity as demand fluctuates, as well as detecting and rebuilding improperly functioning instances. Scalr is also smart enough to know what type of scaling is necessary, but how well it will scale is still a fair question. They perform the scaling activities essentially when a few conditions are met and balance the load uniformly over all instances. Since they don't take the state of the entire system into thought, they can't achieve a global optimal decision.

The Google AppEngine [3] service gives automatic scaling to Web applications. Google App Engine is a Platform as a Service (PaaS) offering that gives you a chance to construct and run applications on Google's infrastructure. With App Engine, there are no servers for you to keep up. You simply transfer your application and it's prepared to go. App Engine applications are easy to build, simple to keep up, and simple to scale as your traffic and data storage needs change. The Google AppEngine service, for instance, requires that the applications be sorted out in such a two tier design and uses the BigTable as its adaptable storage solution. The clients are charged by the CPU cycles consumed, not by the number of application instances. The applications in AppEngine must run inside a sandbox with extreme limitations. It supports basically applications written in Java and Python5 or Google's own Go programming language. This makes it hard to port legacy applications onto their platform.

The aim of the Class Constrained Multiple Knapsack problem (CCMK) [4] is to maximize the total number of packed items in the knapsacks. Each knapsack is restricted by a limited capacity and a bound on the number of different types of items it can hold. It does not try to minimize the number of knapsacks used. It does not support green computing when the system load is low.

The vector bin packing problem looks to minimize the number of bins expected to schedule all n tasks such that the maximum load on any dimension over all bins is limited by a fixed amount. The multi-dimensional constraints have to be considered when packing items into a minimum number of bins. The memory requirement and the CPU demand of an Internet application are considered as individual components in the vector and use vector bin packing to solve our problem. Shockingly, the memory requirement of Internet applications must be fulfilled as a whole: a significant portion of the memory is consumed anyway even when the application gets little load. This is particularly valid for Java applications whose memory utilization may rely on upon the past load because of garbage collection. Thus, the memory requirement can't be separated and fulfill it in a piecemeal manner across the servers. None of the current bin packing problems can be applied in our environment [5].

Application placement in business environments has been studied in [6]-[7]. They run numerous applications on the same set of servers specifically without utilizing VMs or Sandbox. Their method is suitable when the applications are trustworthy. It is not suitable for a cloud environment where applications come from untrusted clients. Their decision algorithm has no concern on green computing and is based on a set of heuristics with no provable limits or optimality.

III. PROPOSED APPROACH

The existing CCBP algorithm mainly focuses on the high demand satisfaction ratio and energy consumption. The performance of this system can be increased by using a two-level hierarchical scheduling model with a Global Centralized Scheduling Center (GCSC) at higher level (to manage load across the clusters) and the Local Centralized Scheduling Center (LCSC) at the lower level (to manage load within a cluster). The Global Centralized Scheduling Center uses the agent based adaptive balancing. The Local Centralized Scheduling Center uses the CCBP algorithm to balance the load. The servers are considered to be heterogeneous with non-uniform capacity. We can divide multiple generations of hardware in a datacenter into "equivalence classes" (server clusters). The proposed system 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. When the datacenter receives a new request for service it queries Global Centralized Scheduler Center for allocation of virtual machine. The Global Centralized Scheduler Center collects the load information from every Local Centralized Scheduler Center and the load will be transferred to the appropriate Local Centralized Schedulers Center. The Global Centralized Scheduler Center also takes over the tasks which offered by the user and return the results of the scheduling to the user.

The Global Centralized Scheduler Center contains an adaptive logic, which checks the state of the servers at regular intervals and independently of the configured weighting. For the extremely powerful "agent based adaptive balancing" method the Global Centralized Scheduling Center periodically checks the system load on all the servers in the farm: Each server machine should provide a file that contains a numeric value in the range between 0 and 99 representing the actual load on this server (0 = idle, 99 = overload,101 = failed,102 = administratively disabled). It is the server's job to provide the actual load in the ASCII file. There are no prerequisites, though, how the servers evaluate this information. Two different strategies are applied, depending on the overall load of the server farm: During normal operation the scheduling algorithm calculates a weighting ratio out of the collected load values and distributes the connections according to it. So if excessive overloading of a server occurs, the weighting is readjusted transparently by the system. As with the weighted round robin, incorrect distribution can then be countered by assigning different weights to the servers available. During a period of very low traffic, however, the load values as reported by the servers will not build a representative sample. A load distribution based on these values would result in uncontrolled, oscillating directives. Therefore in such a situation it is more reasonable, to calculate the load distribution based on the static weight ratio. The Global Centralized Scheduler Center switches to the weighted round robin method automatically when the load on all servers falls below a limit defined by the administrator. If the load rises above the limit the Global Centralized Scheduler Center switches back to the adaptive method.

The traffic is scheduled using the weighted round robin method. The weights used by the weighted round robin method are calculated using the response times from healthcheck requests. Each healthcheck request is timed to see how long it takes to respond. Please note that is assumed that the speed of the healthcheck depends on how slow the machine is which may not always

be the case. The total response times of all the real servers on the Virtual Service are added together, from which the weight of the individual Real Server is calculated. The weights are recalculated approximately every fifteen seconds.

This method balances out the weakness of the simple round robin: Incoming requests are distributed across the cluster in a sequential manner, while taking account of a static "weighting" that can be pre-assigned per server. The administrator simply defines the capacities of the servers available by weighting the servers. The most efficient server A, for example, is given the weighting 100, whilst a much less powerful server B is weighted at 50. This means that Server A would always receive two consecutive requests before Server B receives its first one, and so on. The structure of two level Hierarchical scheduler is given in the fig 3.1.

A. ***GCS Algorithm***

The GCS Algorithm is executed as follows:

(1) Initialize the task parameters;
(2) establish communication with each LCS;
(3) Collect the real load information of each LCS;
(4) calculate the size of load information for each LCS;
(5) If the load information of LCS does not changes or changes within a tolerable range, then do nothing;
(6) Otherwise, update the LCS load information table;
(7) select the LCS with lowest response time and send the task to it;
(8) Check whether there is a request task has been completed, if so, then update requests task table; otherwise, do nothing.

The local centralized scheduling center uses the Class Constrained Bin Packing algorithm to balance the load. The goal is to 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. The system is modelled using 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. Here each application instance is encapsulated inside a virtual machine (VM) and use virtualization technology to provide fault isolation. The local centralized scheduler collects the load information from the computing nodes in that area and balances the load in that local area. Different tasks are assigned to different computing nodes when the local centralized schedulers received request from the global centralized scheduler. The local centralized schedulers also collect the results from every computing node and then send to the global centralized scheduler.

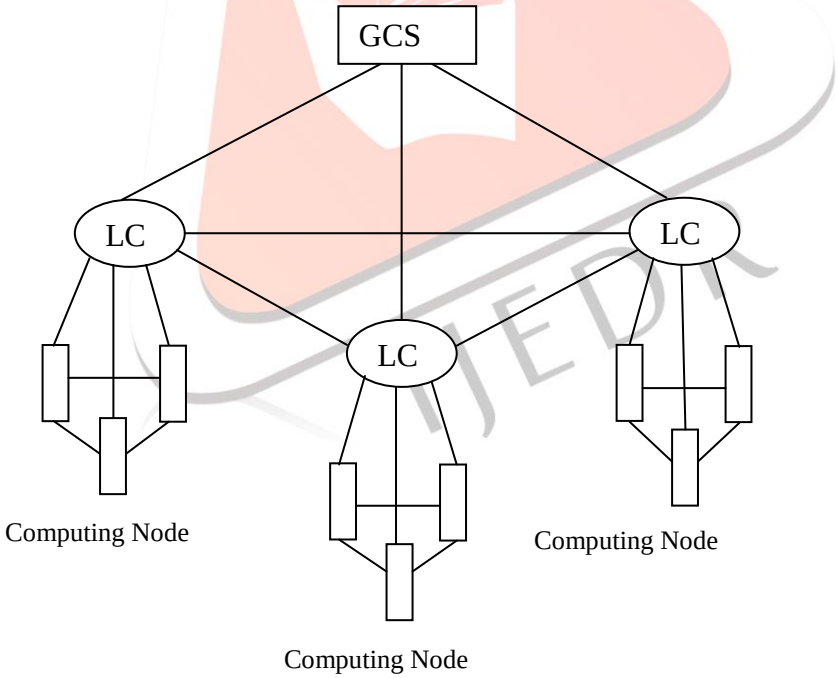

Fig. 3.1: Structure of two level Hierarchical Scheduler.

The local centralized scheduling center uses the CCBP algorithm to balance the load. The class constrained version of this problem isolates the items into classes. Every bin has limit v and can accommodate items from at most c unique classes. It is "class constrained" because of the fact that the class diversity of items packed into the same bin is constrained. The objective is to pack the items into a minimum number of bins. 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 relates to the start of another application occurrence which is a costly operation. Rather, our algorithm attempts to make space for the new item in a currently full bin by moving some of its items into the unfilled bin. Our departure algorithm works as follows. In the event that the color set

does not have an unfilled bin, we can remove any item of that color and the subsequent bin turns into the unfilled bin. Otherwise, if the unfilled bin contains the leaving color, a corresponding item there can be evacuated. In all different cases, we have to remove an item from a present full bin and afterward fill the gap with an item moved in from some other place.

B. LCS Algorithm

The LCS Algorithm is executed as follows:

(1) If the capacity of current bin is reached then open the next bin for packing
(2) Else pack items into current bin
(3) If unfilled bin does not contain color already then make room in currently full bin by shifting items into unfilled bin.

IV. SIMULATION

This section evaluates the performance of our application scheduling algorithm in cloudsim. The input to our simulation includes the resource capacity of each server, the memory demand of each application instances, and the total CPU demand of each application in every decision interval. We define the "demand ratio" as the ratio between the aggregate demand of all applications and the aggregate capacity of all servers. We define a server as active if it has at least one application instance running. We increase the demand ratio (D) from 0.05 to 0.99. The result shows that the number of active servers increases with the demand and reaches 100% when the demand is very high. It decreases slightly with the class constraint when the demand is low, but the effect is small. Thus we conclude that the decision time increases with the demand ratio and with the class constraint.

We evaluate the scalability of the algorithm by increasing both the number of servers and the number of applications. our algorithm is extremely fast, the decision time is less than 4 seconds when the system size is large. The demand satisfaction ratio is independent of the system size. The satisfaction ratio is little affected by the number of applications. When the demand is high, more applications lead to more placement changes. This is because for a given class constraint, more applications means fewer running instances for each application. A larger class constraint helps reduce placement churns. Both of them have a much smaller impact when the demand is lower.

V. CONCLUSION & FUTURE WORK

In this paper, we proposed a two level hierarchical load balancing model for cloud with a Global Centralized Scheduling Center (GCSC) at higher level and the Local Centralized Scheduling Center (LCSC) at lower level. The global centralized scheduling center uses the agent based adaptive balancing to balance the load. The weighted round robin is used to achieve high performance. The weights utilized by the weighted round robin system are figured utilizing the response times from healthcheck demands. The local centralized scheduling center uses the CCBP algorithm to balance the load. It can scale up and down the number of application instances automatically. It achieves high satisfaction ratio of application demand. It saves energy by reducing the number of running instances when the load is low. The CCBP algorithm is used for energy efficient load balancing and the agent based adaptive balancing algorithm is used to increase the performance.

The future work is to give multiple levels of services to the clients while considering fairness when allocating the resources. The premium clients may be given a higher demand satisfaction ratio than other clients, at the point when the resources become tight.

REFERENCES

[1] Zhen Xiao; Qi Chen; Haipeng Luo, "Automatic Scaling of Internet Applications for Cloud Computing Services," Computers, IEEE Transactions on , vol.63, no.5, pp.1111,1123, May 2014.
[2] Scalr: The Auto Scaling Open Source Amazon EC2 Effort. https://www.scalr.net/. Accessed on May 10, 2012.
[3] Google App Engine. http://code.google.com/ appengine/. Accessed on May 10, 2012.
[4] H. Shachnai and T. Tamir, "On two class-constrained versions of the multiple knapsack problem," Algorithmica, vol. 29, no. 3, pp. 442–467, 2001.
[5] C. Chekuri and S. Khanna, "On multidimensional packing problems," SIAM J. Comput., vol. 33, no. 1, pp. 837–851, 2004.
[6] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in Proc. Int. World Wide Web Conf. (WWW'06), May 2006, pp. 595–604.
[7] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in Proc. Int. World Wide Web Conf. (WWW'07), May 2007, pp. 331–340.

Author Profile

Vipin T K received the bachelor's degree from Calicut University, Kerala, India, in 2012. He is currently a Mtech student at Calicut University, Kerala, India. His current research interests include cloud computing and Mobile computing.

Sindhu S received the PhD degree from Anna University, Tamil Nadu, India, in 2013. She is a professor with the Department of Computer Science, NSS College of Engineering, Palakkad, Kerala, India. Her research interests include cloud computing.