Implementation of Server Resource Management System for Cloud Environment

Keerti Priya K¹, Prof. Ashwini²

Student, M.Tech (Software Engineering), NHCE, Bangalore, India¹
Assistant Professor, Information Science & Engineering Department, NHCE, Bangalore, India²

keertipriya.k@gmail.com

Abstract—Cloud computing allows business customers to scale up and down their resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. In existing, we develop a resource allocation system that can avoid overloading in the system effectively while minimizing the number of servers used. Also we introduce the concept of “skewness” to measure the uneven utilization of a server. By minimizing skewness, we can improve the overall utilization of servers in the face of multi-dimensional resource constraints. We propose designing a load prediction algorithm that can capture the future resource usages of applications accurately without looking inside the VMs. The algorithm can capture the rising trend of resource usage patterns and help reduce the placement churn significantly. We present the design and implementation of an automated resource management system that achieves a good balance between the two goals. Two goals are overload avoidance and green computing. We develop a set of heuristics that prevent overload in the system effectively while saving energy used. The experiment results demonstrate that our algorithm achieves good performance.

Keywords: Cloud computing, Dynamic Resource Allocation, Virtual machine, Data center

I. INTRODUCTION

Cloud computing can be the delivery of computing services over a proprietary network or the Internet. These services basically include infrastructures which can be servers, storage devices, etc. It includes software applications as well as development platforms. The Cloud points out to multiple data centers located anywhere in the world that house the hardware which is necessary to offer cloud services. Cloud computing basically collects services and resources which are needed to perform functions as well as to meet dynamically changing needs. The platform of cloud computing always guarantees subscribes that it sticks to the service
level agreement by providing needs and by providing resources as service. However, needs of daily subscribers’ are increasing for computing resources and their needs have platform irrelevance and dynamic heterogeneity. But in the cloud computing environment, when resources are shared and if they are not properly distributed then it will result into wastage of resources. Another important use of cloud computing platform is, it is capable to balance the load amongst several servers dynamically just to avoid hotspots and improve utilization of resources. Hence, the main problems to be solved are how to manage the resources efficiently. In the platform of cloud, resource allocation can take place in two places:

- In the cloud as soon as the application is uploaded, the load balancer gives physical computer with their instances which they have requested for, in order to attempt balancing the computational load of multiple applications across the physical computers.
- When multiple incoming requests are received by an application, each request should be assigned to a particular application instance in order to balance the computational load across a batch of instances of the same application.

The lack of direct capital investment provided by cloud computing is appealing to several businesses. There is a vast discussion on the advantages of cloud model and cloud model cost and on how to move old and traditional applications onto the cloud platform. Here we have a problem: how can a cloud service provider multiplex its virtual resources onto the physical hardware in a best way? This is very much important to be considered because much of the major profits in the cloud model come from this kind of multiplexing. In the research it is found that servers in the existing data centers are mostly under-utilized due to excess provisioning for the peak demand [1] [2]. The cloud model over comes this disadvantage by offering automatic scale up and down in response to variation of load. Along with reducing the cost of the hardware, it also saves a lot of electricity which contributes to a considerable portion of the operational expenses in big data centers.

II. PROBLEM STATEMENT AND SOLUTION

A policy issue exists regarding how to decide the mapping effectively and adaptively so that the demand by resource for VMs is met while the usage of PMs is minimized. This is difficult when the resource needs VMs which are heterogeneous because of the different set of applications they run and change with time whenever the workloads increase and shrink. The main disadvantages here are green computing and overload. The solution proposed is, implementation and design of the automated resource management system is shown that achieves the two goals. Two goals which are mentioned are green computing and overload avoidance.

- Overload avoidance: The capacity of a Physical Memory should be sufficient to satisfy the resource needs of all Virtual Machines running on it.
- Green computing: The number of Physical Memories used should be minimized as long as they can still satisfy the needs of all Virtual Machines

III. THEORITICAL BACKGROUND

A. Skewness Algorithm

Introducing the concept of skewness that quantifies the unevenness in the fulfilment of multiple resources on a server. Let \( n \) be the number of resources we consider and \( r_i \) be the fulfilment of the \( i \)-th resource. Then defining the resource skewness of a server \( p \) as

\[
\text{skewness}(p) = \sqrt{\sum_{i=1}^{n} \left( \frac{r_i}{r} - 1 \right)^2}
\]

Where \( r \) = average fulfilment of all resources for server \( p \).

- Hot and cold spots: When the algorithm executes regularly to check the resource allocation status based on the assumed future resource requirements of the Virtual Machines, then assign a server as a hot spot if the fulfilment of any of its resources is
a hot threshold. This shows that the server is burdened and hence some of the Virtual Machines running on it must be moved away. Then defining a temperature for a hot spot p, as it the square sum of all its resource fulfilment more than the hot threshold:

\[
\text{temperature}(p) = \sum_{r \in R} (r - r_t)^2
\]

R = is the set of resources which are overloaded in server p and \( r_t \) is the hot threshold for resource r.

- **Hot spot mitigation**: Take the sorted list of hot spots in the system in decreasing temperature. Then the goal is to remove all hot spots if possible. Else keeping their temperature as less as possible. For every server \( p \), let’s first decide which of its Virtual Machines should be moved away. Then sort its list of Virtual Machines based on the resulting temperature of the spot server if that Virtual Machine is moved away. We aim to move away the Virtual Machine that can reduce the server’s temperature the highest. In case of ties, we select the Virtual Machine after removal can decrease the skewness of the resource the highest. For each Virtual Machine in the list, see if it can find a destination server to fit it. The server should not become a hot spot after taking this Virtual Machine. Among all such servers, select one whose skewness can be decrease the most by accepting this Virtual Machine. This reduction can be negative that means after the selection of the server whose skewness increases the least. If a destination server is found, note down the migration of the Virtual Machine to that server and update the predicted load of related servers. Else, move on to the next Virtual Machine in the list and try to search a destination server for it.

- **Green computing**: The green computing algorithm is turn on when the average fulfilments of all resources on active servers are less than the green computing threshold. Sorting the list of cold spots in the given system depends on the increasing order of the memory size. Since the need to move away all its Virtual Machines before it can shut down an under-utilized server, define the memory size of a cold spot as the mixer of memory size of all Virtual Machines running on it. For a cold spot \( p \) check if it can move all its Virtual Machines to the destination server. For each Virtual Machine on \( p \), try to find a destination server to that it can accommodate the most. The resource fulfilments of the server after accepting the Virtual Machine must be less than the warm threshold. When it can save energy by considering the under-utilized servers, repeating it can create a lot of hot spots in the future. The warm threshold is made to avoid that. If multiple servers satisfy the above condition, prefer one that is not a currently a cold spot. The reason is that increasing load on a cold spot decreases the likelihood that it can be removed. But it should accept a cold spot as the destination server if needed. Everything that being equal should be selected as a destination server whose skewness can be decreased the most by accepting this Virtual Machine. If it can find the destination servers for every Virtual Machines on a cold spot, then note down the order of migrations and update that is anticipated as a load of similar servers. Else, there is no requirement for not migrate any of its Virtual Machines. Note that we removing cold spots in the system only when the average load of all active servers is below the green computing threshold. Otherwise, leave those cold spots there as potential destination machines for future offloading.

IV. **SYSTEM ARCHITECTURE**
Assume we have ‘n’ number of Virtual Machines (VM1, VM2….), there are some tasks running on each virtual machines. We have data centers, where skewness calculation is done. Skewness calculator calculates the CPU usage, Memory usage and Network usage in terms of percentage. There is a scheduler (Broker). This scheduler will categorize all the virtual machines based on their load. Load in terms of CPU usage, Memory usage and Network usage. The Virtual Machines are segregated into 3 categories: cold spot, warm spot and hot spot. Mitigation section identifies and transfers the virtual machines from hotspot to warm spot and cold spot. It also predicts future in such a way that the Virtual Machines should not enter into hotspot zone after migration of tasks. Broker communicates with data center and informs data center about which Virtual Machine task is going to be migrated to which Virtual Machine.

V. IMPLEMENTATION

A. Language used for implementation

In this project, for implementation purpose Java is chosen as the programming language. Few reasons for which Java is selected as a programming language because, it is platform independent, it is object oriented, it also has a slandered library. Java has a good applet interface and important feature is, it supports swings.

B. Type of Testing

- **Bottom-up Integration**

  This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from bottom to up, processing required for modules subordinate to a given level is always available. Therefore in this case the need for stubs is eliminated. The following integration testing table shows the functions that were combined into different classes and the class as a whole tested for its functionality. This is important to check for error-free interaction between various classes, and maintenance of data integrity.

<table>
<thead>
<tr>
<th>Classes integrated</th>
<th>Functions integrated in each class</th>
<th>Tests done</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class:Main</td>
<td><code>createCloud()</code> <code>executeJob()</code></td>
<td>Class tested to check whether all commands that were applied are working correctly and appropriately or not.</td>
<td>Success</td>
</tr>
<tr>
<td>Class:Broker</td>
<td><code>init()</code> <code>classifyMitigate()</code> <code>mitigateHotSpot()</code> <code>schedulingTask()</code></td>
<td>Class tested to check whether all commands that were applied are working correctly and appropriately or not.</td>
<td>Success</td>
</tr>
<tr>
<td>Class:Data Center</td>
<td><code>submitJob()</code> <code>migrateJob()</code> <code>measureSkewness()</code></td>
<td>Class tested to check whether all commands that were applied are working correctly and appropriately or not.</td>
<td>Success</td>
</tr>
</tbody>
</table>

Table:- Integration testing table

- **Validation Testing**

  At the culmination of integration testing, software is completed and assembled as a package. Interfacing errors are uncovered and corrected. Validation testing can be defined in many ways. Here the testing validates the software function in a manner that is reasonably expected by the customer.
VI. RESULT

The following snapshots define the results or outputs that we will get after step by step execution of all the modules of the system.

<table>
<thead>
<tr>
<th>Functionality to be tested</th>
<th>Input</th>
<th>Tests done</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working of Front-End</td>
<td>User interaction with help of a mouse and keyboard</td>
<td>Appropriate forms open when buttons are clicked</td>
<td>Success</td>
</tr>
<tr>
<td>Working of Job Execution</td>
<td>User has to submit the job and migrate it later calculate the skewness</td>
<td>Job Creation done accordingly with the skewness, hot spot and cold spot calculation.</td>
<td>Success</td>
</tr>
<tr>
<td>Working of Broker</td>
<td>User initialize the cloud them mitigate the hotspot by considering overloaded resources</td>
<td>User classifies the mitigation and Cloud initialized with the hotspot mitigation.</td>
<td>Success</td>
</tr>
<tr>
<td>Working of view graph</td>
<td>User runs simulation and click view graph</td>
<td>Graph of Average Decision Time and Average Migration is displayed</td>
<td>Success</td>
</tr>
</tbody>
</table>

In figure 1, Scheduler Listen Port is the port no. User can define any port no based on his usage. In figure 2, Scheduler IP is the place where we give the IP address of the server in which scheduler is running.
Figure 3 snapshot is used for testing purpose. Here we can define any number of virtual machines and click on start VM. It will automatically generate that many numbers of virtual machines. Figure 4 shows the details of generated virtual machines.

In figure 6, which VM is been migrating to which virtual machine is shown.
Figure 9 shows the graphical image on Average Decision Time. Figure 10 shows the graphs for Average Migration time.
VII. CONCLUSION & FUTURE SCOPE

This paper presents the implementation, evaluation and design of a system for resource management for cloud computing services. Based on the changing demand this system multiplexes virtual to physical resources adaptively. This method uses skewness metric to combine Virtual Machines with different characteristics of resources appropriately so that the capacities of servers are very well utilized. This algorithm achieves both green computing and avoidance of overload for systems with multi-resource constraints.

REFERENCES


© 2015, IJCSMC All Rights Reserved 752