Dynamic Strategies to Stabilize Jobs in Partitioned Public Cloud

DHANU MUKESH[1], G. LAKSHMI NARAYANA[2]
[1]PG Scholar, Dept of CSE, Annamacharya Institute of Technology and Sciences, JNTUA, Tirupathi, AP, India,
E-mail: mukeshdhandu8@gmail.com.
[2]Asst Prof, Dept of CSE, Annamacharya Institute of Technology and Sciences, JNTUA, Tirupathi, AP, India,
E-mail: lakshminarayana0526@yahoo.com.

Abstract: Cloud becomes an emerging technology to expand their services on worldwide, to maintain this craze they are appending new technologies and apps to it. With these updates it was facing some technical challenging issues like scalability, reliability, efficiency and performance to maintain on it. Among these issues load balancing is key challenging job to share a single resource to multiple on demand requested users to satisfy them. Especially in public cloud balancing factor is low because more no of users accessing these freely. To overcome these we proposed an algorithm called game theory to improve efficiency and performance potentiality of public cloud.

Keywords: Load Balancing, Public Cloud, Cloud Partition, Game Theory.

I. INTRODUCTION

Cloud computing is an attracting technology in the field of computer science. In Gartner's report, it says that the cloud will bring changes to the IT industry. The cloud is changing our life by providing users with new types of services. Users get service from a cloud without paying attention to the details. NIST gave a definition of cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. More and more people pay attention to cloud computing. Cloud computing is efficient and scalable but maintaining the stability of processing so many jobs in the cloud computing environment is a very complex problem with load balancing receiving much attention for researchers. Since the job arrival pattern is not predictable and the capacities of each node in the cloud differ, for load balancing problem, workload control is crucial to improve system performance and maintain stability.

Load balancing schemes depending on whether the system dynamics are important can be either static or dynamic. Static schemes do not use the system information and are less complex while dynamic schemes will bring additional costs for the system but can change as the system status changes. A dynamic scheme is used here for its flexibility. The model has a main controller and balancers to gather and analyze the information. Thus, the dynamic control has little influence on the other working nodes. The system status then provides a basis for choosing the right load balancing strategy. The load balancing model given in this article is aimed at the public cloud which has numerous nodes with distributed computing resources in many different geographic locations. Thus, this model divides the public cloud into several cloud partitions. When the environment is very large and complex, these divisions simplify the load balancing. The cloud has a main controller that chooses the suitable partitions for arriving jobs while the balancer for each cloud partition chooses the best load balancing strategy.

II. RELATED WORK

There have been many studies of load balancing for the cloud environment. Load balancing in cloud computing was described in a white paper written by Adler who introduced the tools and techniques commonly used for load balancing in the cloud. However, load balancing in the cloud is still a new problem that needs new architectures to adapt to many changes. Chaczko et al. described the role that load balancing plays in improving the performance and maintaining stability. There are many load balancing algorithms, such as Round Robin, Equally Spread Current Execution Algorithm, and Ant Colony algorithm. Nishant et al. used the ant colony optimization method in nodes load balancing gave a compared analysis of some algorithms in cloud computing by checking the performance time and cost. They concluded that the ESCE algorithm and throttled algorithm are better than the Round Robin algorithm. Some of the classical loads balancing methods are similar to the allocation method in the operating system, for example, the Round Robin algorithm and the First Come First Served (FCFS) rules. The Round Robin algorithm is used here because it is fairly simple.

III. SYSTEM MODEL

There are several cloud computing categories with this work focused on a public cloud. A public cloud is based on the standard cloud computing model, with service provided by a service provider. A large public cloud will include many nodes and the nodes in different geographical locations. Cloud partitioning is used to manage this large cloud. A cloud partition is a subarea of the public cloud with divisions based on the geographic locations. The architecture is shown in Fig.1. The load balancing strategy is based on the cloud partitioning concept. After creating the cloud partitions, the load balancing then starts, when a job arrives at the system, with the main controller deciding which cloud partition should receive the job. The partition load balancer then decides how to assign the jobs to the nodes. When the load status of a cloud partition is normal, this partitioning can be accomplished locally. If the cloud partition load status is not normal, this job should be transferred to another partition.

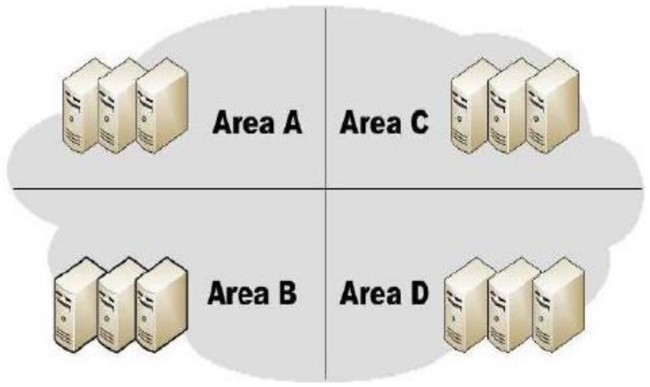

Fig.1. Typical cloud partitioning.

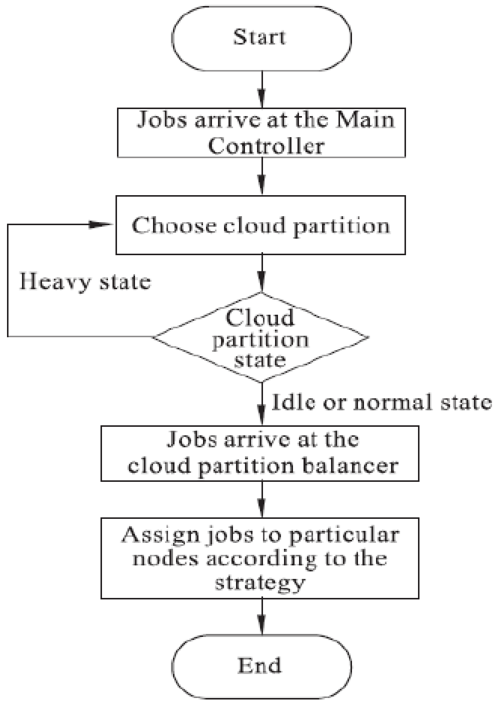
Start
Jobs arrive at the Main Controller
Choose cloud partition
Heavy state
Cloud partition state
Idle or normal state
Jobs arrive at the cloud partition balancer
Assign jobs to particular nodes according to the strategy
End

Fig.2. Job assignment strategy Main controller and balancers.

The load balance solution is done by the main controller and the balancers. The main controller first assigns jobs to the suitable cloud partition and then communicates with the balancers in each partition to refresh this status information (see fig 2). Since the main controller deals with information for each partition, smaller data sets will lead to the higher processing rates. The balancers in each partition gather the status information from every node and then choose the right strategy to distribute the jobs. The relationship between the balancers and the main controller is shown in Fig.3.

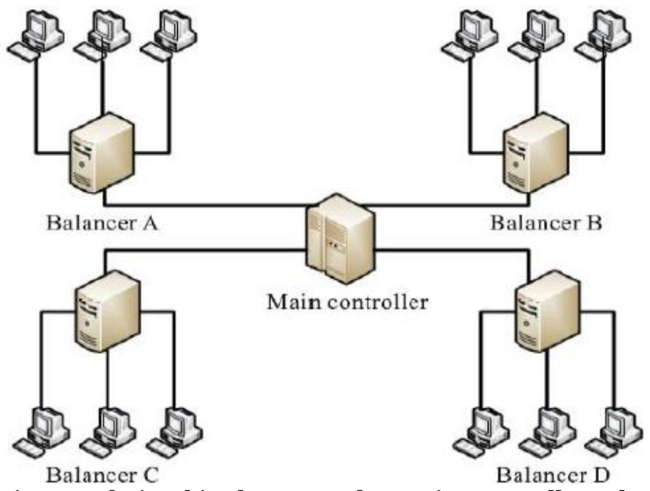

Fig.3. Relationship between the main controller, the balancers and the nodes.

A. Assigning Jobs to the Cloud Partition

When a job arrives at the public cloud, the first step is to choose the right partition. The cloud partition status can be divided into three types:

Idle: When the percentage of idle nodes exceeds α, change to idle status.
Normal: When the percentage of the normal nodes exceeds β, change to normal load status.
Overload: When the percentage of the overloaded nodes exceeds γ, change to overloaded status.

Algorithm: Game Theory

Algorithm 1 Best Partition Searching

```
begin
  while job do
    searchBestPartition (job);
    if partitionState == idle || partitionState == normal then
      Send Job to Partition;
    else
      search for another Partition;
    end if
  end while
end
```

The parameters α, β and γ are set by the cloud partition balancers. The main controller has to communicate with the balancers frequently to refresh the status information. The main controller then dispatches the jobs using the following strategy: When job i arrives at the system, the main controller queries the cloud partition where job is located. If this location's status is idle or normal, the job is handled locally. If not, another cloud partition is found that is not overloaded. The algorithm is shown in Algorithm below.

B. Assigning Jobs to the Nodes in the Cloud Partition

The cloud partition balancer gathers load information from every node to evaluate the cloud partition status. This evaluation of each node's load status is very important. The first task is to define the load degree of each node. The node load degree is related to various static parameters and dynamic parameters. The static parameters include the number of CPU's, the CPU processing speeds, the memory size, etc. Dynamic parameters are the memory utilization ratio, the CPU utilization ratio, the network bandwidth, etc. The load degree is computed from these parameters as below:

Step 1: Define a load parameter set: F = {F1,F2 ,...,Fn } with each Fi(1≤ i ≤ m, Fi[0,1]) parameter being either static or dynamic. m represents the total number of the parameters.

Step 2: Compute the load degree as:

$$\text{Load\_degree(N)} = \sum_{i=1}^{m} \propto iFi \qquad (1)$$

$\propto i \quad (\sum_{i=1}^{m} \propto i = 1)$ are weights that may differ for different kinds of jobs. N represents the current node.

Step 3: Define evaluation benchmarks. Calculate the average cloud partition degree from the node load degree statistics as:

$$\text{Load\_degree}_{(avg)} = \frac{\sum_{i=1}^{m} Load\_degree\,(Ni)}{n} \qquad (2)$$

The bench mark Load_degreeHigh is then set for different situations based on the Load_degreeavg.

Step 4: Three nodes load status levels are then defined as: Idle When

$$\text{Load\_Degree(N)=0} \qquad (3)$$

There is no job being processed by this node so the status is charged to Idle. Normal For

$$0 < \text{Load\_degree(N)} \leq \text{Load\_degree}_{high} \qquad (4)$$

The node is normal and it can process other jobs. Overloaded when the node is not available and cannot receive jobs until it returns to the normal.

$$\text{Load\_degree}_{high} \leq \text{Load\_degree(N)} \qquad (5)$$

The load degree results are input into the Load Status Tables created by the cloud partition balancers. Each as a Load Status Table and refreshes it each fixed period T. The table is then used by the balancers to calculate the partition status. Each partition status has a different load balancing solution. When a job arrives at a cloud partition, the balancer assigns the job to the nodes based on its current load strategy. This strategy is changed by the balancers as the cloud partition status changes.

IV. CLOUD PARTITION LOAD BALANCING STRATEGY

A. Motivation

Good load balance will improve the performance of the entire cloud. However, there is no been developed in improving. Each particular method has advantage common method that can adapt to all possible different situations. Various methods have in a particular area but not in all situations. Therefore, the current model integrates several methods and switches between the load balance methods based on the system status. A relatively simple method can be used for the partition idle state with a more complex method for the normal state. The load balancers then switch methods as the status changes. Here, the idle status uses an improved Round Robin algorithm while the normal status uses a game theory based load balancing strategy.

B. Load Balance Strategy for the Idle Status

When the cloud partition is idle, many computing resources are available and relatively few jobs are arriving. In this situation, this cloud partition has the ability to process jobs as quickly as possible so a simple load balancing method can be used. There are many simple load balance algorithm methods such as the Random algorithm, the Weight Round Robin, and the Dynamic Round Robin. The Round Robin algorithm is used here for its simplicity. The Round Robin algorithm is one of the simplest load balancing algorithms, which passes each new request to the next server in the queue. The algorithm does not record the status of each connection so it has no status information. In the regular Round Robin algorithm, every node has an equal opportunity to be chosen. However, in a public cloud, the configurable and the performance of each node will be not the same; thus, this method may overload some nodes. Thus, an improved Round Robin algorithm is used, which called "Round Robin based on the load degree evaluation".

The algorithm is still fairly simple. Before the Round Robin step, the nodes in the load balancing table are ordered based on the load degree from the lowest to the highest. The system builds a circular queue and walks through the queue again and again. Jobs will then be assigned to nodes with low load degrees. The node order will be changed when the balancer refreshes the Load Status Table. However, there may be read and write inconsistency at the refresh period T. When the balance table is refreshed, at this moment, if a job arrives at the cloud partition, it will bring the inconsistent problem. The system status will have changed but the information will still be old. This may lead to an erroneous load strategy choice and an erroneous nodes order. To resolve this problem, two Load Status Tables should be created as: Load Status Table_1 and Load Status Table_2. A flag is also assigned to each table to indicate Read or Write. When the flag = "Read", then the Round Robin based on the load degree evaluation algorithm is using this table. When the flag = "Write", the table is being refreshed, new information is written into this table. Thus, at each moment, one table gives the correct node locations in the queue for the improved Round Robin algorithm, while the other is being prepared with the updated information. Once the data is refreshed, the table flg is changed to “Read” and the other table’s flag is changed to “Write”. The two tables then alternate to solve the inconsistency. The process is shown in Fig.4.

C. Load Balancing Strategy for the Normal Status

When the cloud partition is normal, jobs are arriving much faster than in the idle state and the situation is far more complex, so a different strategy is used for the load balancing. Each user wants his jobs completed in the shortest time, so the public cloud needs a method that can complete the jobs of all users with reasonable response time. Penmatsa and Chronopoulos proposed a static load balancing strategy based on game theory for distributed systems. And this work provides us with a new review of the load balance problem in the cloud environment. As an implementation of distributed system, the load balancing in the cloud computing environment can be viewed as a game.

Game theory has non-cooperative games and cooperative games. In cooperative games, the--> decision makers eventually come to an agreement which is called a binding agreement. Each decision maker decides by comparing notes with each others. In non-cooperative games, each decision maker makes decisions only for his own benefit. The system then reaches the Nash equilibrium, where each decision maker makes the optimized decision. The Nash equilibrium is when each player in the game has chosen a strategy and no player can benefit by changing his or her strategy while the other player’s strategies remain unchanged.

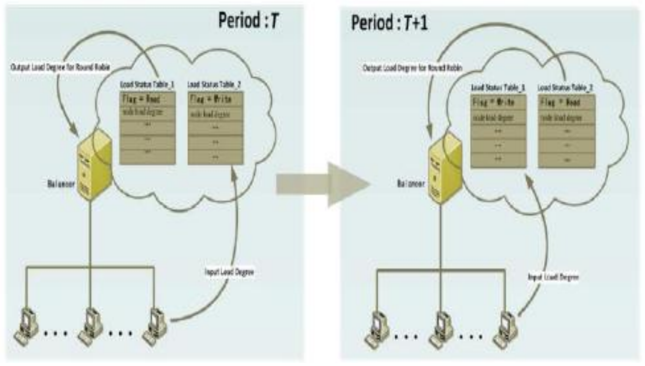

Fig. 4 The solution of inconsistently problem.

There have been many studies in using game theory for the load balancing. Grosu et al. proposed a load balancing strategy based on game theory for the distributed systems as a non-cooperative game using the distributed structure. They compared this algorithm with other traditional methods to show that their algorithm was less complexity with better performance. Aote and Kharat gave a dynamic load balancing model based on game theory. This model is related on the dynamic load status of the system with the users being the decision makers in a non-cooperative game. Since the grid computing and cloud computing environments are also distributed system, these algorithms can also be used in grid computing and cloud computing environments. Previous studies have shown that the load balancing strategy for a cloud partition in the normal load status can be viewed as a non cooperative game, as described here. The players in the game are the nodes and the jobs. Suppose there are n nodes in the current cloud partition with N jobs arriving, then define the following parameters: μi : Processing ability of each node, i= 1,...,n. In this model, the most important step is finding the appropriate value of sij. The current model uses the method of Grosu et al. called “the best reply” to calculate sij. For all nodes. This procedure gives the Nash equilibrium to minimize the response time of each job. The strategy then changes as the node’s status change.

V. FUTURE WORK

Since this work is just a conceptual framework, more work is needed to implement the framework and resolve new problems. Some important points are:

Cloud division rules: Cloud division is not a simple problem. Thus, the framework will need a detailed cloud division methodology. For example, nodes in a cluster may be far from other nodes or there will be some clusters in the same geographic area that are still far apart. The division rule should simply be based on the geographic location (province or state).

How to set the refresh period: In the data statistics analysis, the main controller and the cloud partition balancers need to refresh the information at a fixed period. If the period is too short, the high frequency will influence the system performance. If the period is too long, the information will be too old to make good decision. Thus, tests and statistical tools are needed to set a reasonable refresh period.

A better load status evaluation: A good algorithm is needed to set Load_degreehigh and Load_degreelow, and the evaluation mechanism needs to be more comprehensive.

Find other load balance strategy: Other load balance strategies may provide better results, so tests are needed to compare different strategies. Many tests are needed to guarantee system availability and efficiency.

VI. CONCLUSION

Cloud computing system has widely been adopted by the industry though there are many existing issues like load balancing, migration of virtual machine, server unification which have been not yet fully addressed. Load balancing is the most central issue in the system to distribute load in efficient manner. It also ensures that every computing resource is distributed efficiently and fairly. Existing load balancing technique have been studied mainly focus on reducing overhead, reducing migration time and improving performance.

VII. REFERENCES

[1] B. Adler, Load balancing in the cloud: Tools, tips and techniques, http://www.rightscale.com/ info center/white papers/ Load- Balancing-in-the-Cloud.pdf, 2012.

[2]S. Penmatsa and A. T. Chronopoulos, Game-theoretic static load balancing for distributed systems, Journal of Parallel and Distributed Computing, vol. 71, no. 4, pp. 537-555, Apr. 2011.
[3]D. Grosu, A. T. Chronopoulos, and M. Y. Leung, Load balancing in distributed systems: An approach using cooperative games, in Proc. 16th IEEE Intl. Parallel and Distributed Processing Symp., Florida, USA, Apr. 2002, pp. 52-61.
[4] S. Aote and M. U. Kharat, A game-theoretic model for dynamic load balancing in distributed systems, in Proc.The International Conference on Advances in Computing, Communication and Control (ICAC3 '09), New York, USA, 2009, pp.235-238.
[5] R. Hunter, The why of cloud, http://www.gartner.com/ Display Document?doc cd=226469&ref= g nor eg, 2012.
[6]Gaochao Xu et al.: A Load Balancing Model based on Cloud Partitioning for the public Cloud.

Author's Profile:

Dhandu Mukesh received the B.Tech Degree in Information Technology from Annamacharya Institute of Technology and Sciences, University of JNTUA in 2012.He is currently working towards the Master's Degree in Computer Science and Engineering, in Annamacharya Institute of Technology and Sciences University of JNTUA. His interest lies in the areas of Cloud Computing, DBMS.

G. Lakshmi Narayana Received B.Tech and M.Tech Degrees in Computer Science and Engineering from Narayana engineering college Nellore JNTUA, SV University in 2003 & 2007 respectively. Currently he is a Assistant Professor in the Department of Computer Science and Engineering at AITS-Tirupati. He has published a paper titled "Channel fading using camov in MANETS hand-off Strategy". in Journals and refereed Conference Proceedings. His current interests include Data Mining and applications, Operating Systems, Compiler Design and Information Security.