A LOAD BALANCING MODEL BASED ON CLOUD PARTITIONING

[1]A.Vimal, [2]C.Sivakumar

[1]Research Scholar, Department of Information Technology, Jayam College of Engineering and Technology, Dharmapuri

[2]Associate Professor, Department of Information Technology, Jayam College of Engineering and Technology, Dharmapuri

[1]vimalcse117@rediffmail.com , [2]svkumar650@gmail.com

Abstract: In this paper we present a game theoretic framework for obtaining a user-optimal load balancing scheme in hetero- generous distributed systems. Load balancing in the cloud computing environment has an important impact on the performance. Good load balancing makes cloud computing more efficient and improves user satisfaction. This article introduces a better load balance model for the public cloud based on the cloud partitioning concept with a switch mechanism to choose different strategies for different situations. The algorithm applies the game theory to the load balancing strategy to improve the efficiency in the public cloud environment.

Keywords: Game theory, load balancing, cloud computing.

1. INTRODUCTION

Cloud computing is an attracting technology in the field of computer science. In Gartner's report [1], 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' distributed system can be viewed as a collection of computing and communication resources shared by active users. When the demand for computing power increases the load balancing problem becomes important. A general for- mulation of this problem is as follows: given a large number of jobs, find the allocation of jobs to computers optimizing a given objective function (e.g. total execution time).

There are three typical approaches to load balancing problem in distributed systems:

- **Global approach:** In this case there is only one decision maker that optimizes the response time of the entire system over all jobs and the operating point is called social (overall) optimum.
- Cooperative approach: In this case there are several decision makers (e.g. jobs, computers) that cooperate in making the decisions such that each of them will operate at its optimum. Decision makers have complete freedom of pre play communication to make joint agreements about their operating points. This situation can be modeled as a cooperative game and game theory offers a suitable modeling frame work.
- Non cooperative approach: In this case there are several decision makers (e.g. users, jobs) that are not allowed to cooperate in making decisions. Each decision maker op- times its own response time independently of the others and they all eventually reach equilibrium. This situation can be viewed as a non-cooperative game among decision makers. The equilibrium is called Nash equilibrium and it can be obtained by a distributed non cooperative policy. At the Nash equilibrium a decision maker cannot receive any further benefit by changing its own decision. If the number of decision makers is not finite the Nash equilibrium is called War drop equilibrium.

1.1 Existing Result

There exist only few studies on game theoretic models and algorithms for load balancing in distributed systems. Kameda et al. [6] studied non cooperative games and de- rived load balancing algorithms for computing the War drop equilibrium in single class and multi-class job distributed systems. Rough garden [16] formulated the load balancing problem as a Stackelberg game. In this type of non-cooperative game one player acts as a leader and the rest as followers. He showed that it is NP-hard to compute the optimal Stackelberg strategy and presents efficient algorithms to compute strategies inducing near-optimal solutions.

Routing traffic in networks is a closely related problem that received more attention. Orda et al. [14] studied a non- cooperative game in a network of parallel links with convex cost functions. They studied the existence and uniqueness of the Nash equilibrium. Altman et al. investigated the same problem in a network of parallel links with linear cost functions. Korilis et al. considered the capacity allocation problem in a network shared by noncooperative users. They studied the structure and the properties of Nash equilibrium for a routing game with M/M/1 type cost functions. An important line of research was initiated by Koutsoupias and Papadimitriou, who considered a non-cooperative routing game and proposed the ratio between the worst possible Nash equilibrium and the overall optimum as a measure of effectiveness of the system. Rough garden and Tardos showed that in a network in which the link cost functions are linear the flow at Nash equilibrium has total latency at most 4/3 that of the overall optimal flow. They also showed that if the link cost functions are assumed to be only continuous and non-decreasing the total latency may be arbitrarily larger than the minimum possible total latency.

1.2 Our results

Most of the previous studies on static load balancing considered as their main objective the minimization of overall expected response time. This is difficult to achieve in distributed systems where there is no central authority control- ling the allocation and users are free to act in a selfish manner. Our goal is to find a formal framework for characterizing user-optimal allocation schemes in distributed systems. The framework was provided by noncooperative game the- ory which has been applied to routing and flow control problems in networks but not to load balancing in distributed systems. Using this framework we formulate the load balancing problem in distributed systems as a noncooperative game among users. The Nash equilibrium provides a user- optimal operation point for the distributed system. We give a characterization of the Nash equilibrium and a distributed algorithm for computing it. We compare the performance of our noncooperative load balancing scheme with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user in the distributed system.

1.3 Organization

The paper is structured as follows. In Section 2 we present the system model and we introduce our load balancing non- cooperative game. In Section 3 we derive a greedy dis- tributed algorithm for computing the Nash equilibrium for our load balancing game. In Section 4 the performance of our load balancing scheme is compared with those of other existing schemes. In Section 5 we draw conclusions and present future directions.

2. LOAD BALANCING AS A N O N COOPERATIVE G A M E AMONG USERS

We consider a distributed system that consists of heterogeneous computers shared by users. Each computer is modeled as an M/M/1 queuing system (i.e. Poisson arrivals and exponentially distributed processing times). Com- puteris characterized by its average processing rate Jobs are generated by user with an average rat, and is the total job arrival rate in the system. The total job arrival rate must be less than the aggregate processing rate of the system model is presented in Figure 1. The users have to decide on how to distribute their jobs to computers such that they will operate optimally. Thus user must find the fraction of all its jobs that are assigned to computer such that the expected execution time of its jobs is minimized.

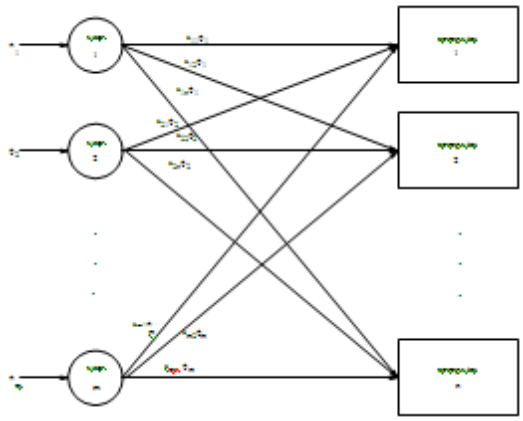

Figure 1: The distributed system model

We formulate this problem as a noncooperative game among users under the assumption that users are 'selfish'. This means that they minimize the expected response time of their own jobs.

3. 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 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.

3.1 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. 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. refresh this status information. 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.

3.2 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 change to idle status. Normal: When the percentage of the normal nodes exceeds change to normal load status. Overload: When the percentage of the overloaded nodes exceeds, change to overloaded status. The parameters and 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 1.

3.3 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

Algorithm 1 Best Partition Searching

begin

while job do

Search Best Partition

(job);

if partition State == idle k partition State == normal

then

Send Job to Partition;

Else

search for another

Partition; end if

end

while

end

each nodes

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 D fF1; F2;; Fmg with each Fi .1 6 i 6 m; Fi 2 .0; 1□/ parameter being either static or dynamic. M represents the total number of the parameters.

Step 2 Compute the load degree as: Load degree. N / D Xm iD1 iFi ; i . Pn iD1 _i D 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: Load degree avg D Pn iD1 Load degree.Ni / nThe bench mark Load degree high is then set for different situations based on the Load degreeavg.

Step 4 Three nodes load status levels are then defined as: Idle When Load degree.N / D 0;there is no job being processed by this node so the status is charged to Idle. Normal For 0 < Load degree.N / 6 Load degreehigh; the node is normal and it can process other jobs. Overloaded When Load degreehigh 6 Load degree.N /; the node is not available and cannot receive jobs until it returns to the normal. The load degree results are input into the Load Status Tables created by the cloud partition balancers. Each balancer has 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.

4. CLOUD PARTITION LOAD BALANCING STRATEGY

4.1 Motivation

Good load balance will improve the performance of the entire cloud. However, there is no common method that can adapt to all possible different situations. Various methods have been developed in improving existing solutions to resolve new problems. Each particular method has advantage in a particular area but not in all situations. Therefore, the current model integrates several methods and switches between the load balance method 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 .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.

4.2 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 configuration 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: 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 flag is changed to "Read" and the other table's flag is changed to "Write". The two tables then alternate to solve the inconsistency.

4.3 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 other's. 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. 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 noncooperative 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: i : Processing ability of each node, i D 1; n. j : Time spending of each job. D PN jD1 _j : Time spent by the entire cloud partition, _ < Pn iD1 _i . sj i : Fraction of job j that assigned to node I Pn iD1 sj i D 1 and 0 6 sj i 6 1). In this model, the most important step is finding the appropriate value of sj i . The current model uses the method of Grosu et al. called "the best reply" to calculate sj i of each node, with a greedy algorithm then used to calculate sj i 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 statuses change.

5. 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 periods.

A better load status evaluation: A good algorithm is needed to set Load degreehigh and Load degree, 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.

6. ACKNOWLEDGEMENTS

We would like to thank the editors and anonymous reviewers for their valuable comments and helpful suggestions.

REFERENCES

[1] R.Hunter, The why ofcloud ,http://www.gartner.com/DisplayDocument?doc cd=226469&ref= g noreg, 2012.

[2] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, Cloud computing: Distributed internet computing for IT and scientific research, Internet Computing, vol.13, no.5, pp.10- 13, Sept.-Oct. 2009.

[3] P. Mell and T. Grance, The NIST definition of cloud computing, http://csrc.nist.gov/ publications/nistpubs/800- 145/SP800-145.pdf, 2012.

[4] Microsoft Academic Research, Cloud computing,http://libra.msra.cn/Keyword/6051/cloud computing?query= cloud%20computing, 2012.

[5] Google Trends, Cloud computing, http://www.google.com/trends/explore#q=cloud%20com puting, 2012.

[6] N. G. Shivaratri, P. Krueger, and M. Singhal, Load distributing for locally distributed systems, Computer, vol. 25, no. 12, pp. 33-44, Dec. 1992.

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

[8] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, and C. Mcdermid, Availability and load balancing in cloud computing, presented at the 2011 International Conference on Computer and Software Modeling, Singapore, 2011.

[9] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, N. Nitin, and R. Rastogi, Load balancing of nodes in cloud using ant colony optimization, in Proc. 14th International Conference on Computer Modelling and Simulation (UKSim), Cambridgeshire, United Kingdom,Mar. 2012, pp. 28-30.

[10] M. Randles, D. Lamb, and A. Taleb-Bendiab, A comparative study into distributed load balancing algorithms for cloud computing, in Proc. IEEE 24th International Conference on Advanced Information Networking and Applications, Perth, Australia, 2010, pp. 551-556.

[11] A. Rouse, Public cloud,http:// search cloud computing. Tech target.com/definition/public- cloud, 2012.

[12] D. MacVittie, Intro to load balancing for developers The algorithms, https://devcentral.f5.com/blogs/us/introtoload- balancing -for-developers -ndash-thealgorithms,2012.

[13] 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.

[14] 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.

[15] 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.