Token based Load Balancing Strategy in Distributed Systems

Ankita Singhal[1], Anuj Tiwari[2] ,Archana Nigam[3]

[1,3] *M.Tech IIT Roorkee INDIA,*[2]*M.Tech DAVV Indore INDIA,*

Abstract - **Load balancing in a distributed system is the process of redistributing the workload among various nodes so as to improve resource utilization and the mean response time and also to balance the workload among the nodes of the system to avoid the situation in which one node is overloaded while other is sitting idle. A dynamic load balancing approach needs no prior knowledge about the global status of the distributed system and does balancing based on the current status of the system. Most of the techniques involve communication between the nodes to exchange their load information to make load balancing decisions i.e. where the arrived task can be best executed from. But this considerably increases mean response time.**

This paper presents a token based technique for load balancing in which there is no communication among the nodes and so no exchange of load information messages. Each individual node is configured to make its own decision whether to accept the arriving request or not and once accepted it will be executed from there. No other nodes can now accept this request.

Keywords: Loadbalancing, mean response time, task allocation, task transfer, distributed systems

I. INTRODUCTION

A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility. In the Internet, several types of services use replicated server nodes which are geographically dispersed across the whole network. The aim of this approach is to prevent too many accesses from concentrating at a particular node, which causes degradation of the response time of a node itself and congestion in the network around that node. In distributed system, load balancing services distribute client workload equally among various back-end servers (nodes) in order to obtain the best response time possible. Moreover, it should be avoided that some tasks are forced to wait for a very long time. For this, tasks arrived at heavily loaded nodes should be forwarded to lightly loaded nodes. There have been numerous number of techniques proposed for this purpose. The techniques can be divided mainly into two categories – load balancing on the part of router (or proxy) and server side load balancing. The term load balancing is often used as task allocation. This task allocation scheme can have another two broad categories – centralized and distributed. In centralized approach, there is a single dedicated node that performs the task allocation by monitoring over the various parameters of all the other nodes in the system. This approach can be used for small systems since there is a single point of failure in this approach and thus can bring down the whole system to a standstill. In distributed approach, task allocation is performed by all the nodes by communicating with each other. The distributed approach provides a good fault tolerance and scalability but most of the techniques proposed involve a large range of broadcasting of their load information to other nodes which substantially increases the traffic on the network.

This paper implements server side load balancing and the main aim of the technique proposed is to improve the overall response time of arriving task. The proposed work tries to fill up the gaps that were found in existing load balancing strategies. The main aim of existing strategies is that each node must have almost equal load distribution i.e. if there are 4 nodes in a system then each must have near to 25% of total load. But in an attempt to distribute the load in such a disciplined manner, the main aim of improving response time is overlooked. The complexity of the algorithms leads to degradation of response time rather than improving it. The technique proposed in this paper tries to improve the overall response time but does not necessarily distributes the load on the backend servers very evenly. Till a server has tasks much less than its capacity it should readily accept more tasks to run without bothering about the load on other servers. This will save the time of communication with other nodes, exchanging load information with them and then finally deciding where the task will be run. If a server is underloaded it must execute a task immediately.

Proc. of the Intl. Conf. on Advances in Computer, Electronics and Electrical Engineering
Editor In Chief Dr. R. K. Singh. Copyright © 2012 Universal Association of Computer and Electronics Engineers.
All rights reserved. ISBN: 978-981-07-1847-3

SEEK DIGITAL LIBRARY

II. LITERATURE REVIEW

With the great advancements in computer technology and the availability of many distributed systems, the problem of load balancing in distributed systems has gained a higher attention and importance. Task allocation in distributed systems has been studied and many policies proposed [1], [2], [3], [4]. Consequently, a vast amount and variety of research has been conducted in an attempt to solve this problem. There are basically two main strategies for load balancing-static and dynamic. In the static approach, load balancing is achieved by providing a mapping or assignment from a set of tasks to a set of processors such that system's performance is maximised. The chord protocol uses this technique [5]. But in static approach prior knowledge about the global status of the distributed system, job resource requirement, and communication time are assumed. In the dynamic approach e.g. [6], [7], load balancing is based on the current state of the system; tasks are allowed to move dynamically from an overloaded node to an under loaded node to receive faster service. This ability to react to changes in the system is the main advantage of the dynamic approach to load balancing. Although finding a dynamic solution is much more complicated than finding a static one, dynamic load balancing can produce a better performance because it makes load balancing decisions based on the current load of the system. So the task allocation scheme in this paper will use dynamic task allocation scheme.

Task allocation schemes proposed till now typically concentrate on whether to use load information of remote nodes or not. In any typical distributed task allocation each node behaves as follows. In random subset task allocation [8], when a task arrives at the local node, the node decides whether to execute task locally or to transfer it. If the node decides to transfer it, the node selects a subset of remote nodes randomly. Then it selects the node with the lightest load as the destination node. In order to select the destination node, the local node probes all the nodes in the subset to get their load information.

In Nearest Neighbour task allocation [9], since communication is limited to only neighbouring nodes, communication delay is short and hence the load information is not as old as it was in random subset task allocation technique.

III. DETAILED PROBLEM STATEMENT

Our main threshold for comparing all the results will be the NearestNeighbor(NN) approach described in [9]. In NN approach, when a task arrives, the local node always determines whether the task can be executed locally or should be transferred. If the destination node is more appropriate than local node, the task is transferred. Otherwise the task is executed locally. The destination node is selected from its neighbors. The local node selects the neighbor with the lightest node. In NN, each node stores information of all of its neighbors. If the load of a node is changed, i.e. a new task is added to the run queue of the node or a task finished executing, the node sends a message to *all* of its neighbors immediately. Receiving the message, the neighbors update its load information.

In this paper we will assume the group of neighbors but the protocol inside each group will change. Before explaining the protocol that will be used, first let us try to understand the problem in the existing protocol.

The main problem in this approach or any other load balancing approach such as RandomSubset [8] is the "broadcast" of load information. Since the load over any node is subject to frequent changes, a lot of bandwidth is consumed in sending and receiving the 'load information messages '. Another problem with this approach is the time involved in transferring the task. Task transfer is not a simple operation and involves some overhead. This increases processing delay.

So here in this paper, we aim at eliminating the broadcast of load information and decreasing the processing delay involved.

IV. PROPOSED SOLUTION AND ASSUMPTIONS

Assumptions involved:

- All tasks are of same type that is nearly same service time is required by each arriving task.
- All nodes are identical in terms of information.

Here we will first try to minimize processing delay i.e. we will propose a technique that does not involve task transfer and then we will come to first problem (of broadcasting) described above.

To eliminate the task transfer problem we first get at the root of the problem. Why there is a need for task transfers? Task transfer at nodes mainly occurs because the task reaches the wrong node i.e. a node that is already overloaded or does not have minimum delay and so it has to transfer it to other destination node. To keep a check on this parameter we proceed as follows:

Each server has its own capacity beyond which it cannot serve requests. Since each server knows its own capacity, don't let the server, which has reached its capacity; accept any request since if any request reaches such a server it will have to be transferred. When the load on this server goes down it can again become active to accept request. So, if a request has reached any of the server it will be served from there .When all of the nodes are overloaded, request will

SEEK DIGITAL LIBRARY

wait (for a specified amount of time) for any of the servers to get active again.

For implementing this approach, we used the concept of token as used in token ring protocol. The token is a sequence of special bits that is known by each node in the system. There is only one token for all the nodes in the system. As soon as a task arrives, any of the servers that is ready to accept the request (that is length of run queue is less than its capacity) seizes the token. Once the token is seized no other node can seize it since the token is unique. Thus, the task is completed by the node that seized the token and then releases it. When a new task arrives, again the token is seized by any of the nodes that can complete the task, completes it and then releases. This procedure is repeated each time a new task arrives.

Considering this approach we have solved both the problems. Firstly, no task transfer is needed and secondly since all the decisions are made solely on the basis of load of each individual server there is no need to broadcast the load information to other nodes.

We will call the proposed technique as SeizeToken.

V. SIMULATION

In order to determine the behavior of the scheme mentioned in section IV we perform the following simulations:

We simulate the NearestNeighbor and SeizeToken to compare the results. The simulation is performed over 50 concurrent requests. The simulator for this was implemented using Netbeans7.1. For nearest neighbor, time taken in task transfer is 100 msec. The mean task service time is 50 msec. Arrived tasks are sent to randomly selected nodes. Then, they are executed locally or transferred to other node according to task allocation scheme. The number of transfers depends on the number of nodes in the system. We performed the simulation for number of nodes as two and then three. The communication delay between the node and its neighbor is assumed to be fixed.

For SeizeToken, as the technique is proposed no task transfer takes place. The arrived tasks are sent to any node that is ready to accept requests and is served from there. For this also we performed the simulations for two and three nodes.

Simulations were performed for 50 concurrent requests for each case. As performance metrics, we use the mean response time. Mean response time is defined as the time from when the task arrives at the system until the task leaves the system. There are 4 graphs plotted from the data obtained from simulations. Fig 1 is the primary result which shows the difference in mean response time from the existing task transferring load balancing strategies. Fig 2 and Fig 3 shows the effect of varying the number of nodes in the system in two types of strategies. Finally, fig 4 clears the fact that the load, in the proposed technique is not very even.

Simulation Results

Now we will explain each graph in a greater detail. Fig 1 shows the response time taken for each request. As described earlier, 50 concurrent requests were sent over the system and mean response time for each request was calculated. This was done for both NearestNeighbor and SeizeToken algorithms. As can be seen from the graph, SeizeToken algorithm, on an average, takes less response time than NN. Thus the proposed method outperforms the existing technique. One main reason for this improvement is the way in which each arriving task is handled. Any task as it arrives is executed from there only and is not transferred. Thus, SeizeToken seems to be more efficient.

The graph of Fig 2 and Fig 3 gives a comparison of the time taken to execute a task when there are three servers and two servers in NearestNeighbor and SeizeToken. As demonstrated, in NearestNeighbor when there are more number of servers, a task is transferred more number of times than in case of less number of servers thus increasing the response time on an average.

Contrary to this, our proposed technique SeizeToken does not have any effect of number of servers involved, on response time because virtually all servers are working independently and they do not communicate with each other. Thus a task is executed from any of the randomly selected node and is independent of other nodes.

Fig 4 shows the compromise that is done to avoid task transfers. It gives an overview of load on each server at any instant of time. The load distribution in SeizeToken is not very even as same server can seize token again and again to execute a request. The load distribution was found to be more disciplined in case of NN. But this does not affect the response time much and only improves it over NearestNeighbor

SEEK DIGITAL LIBRARY

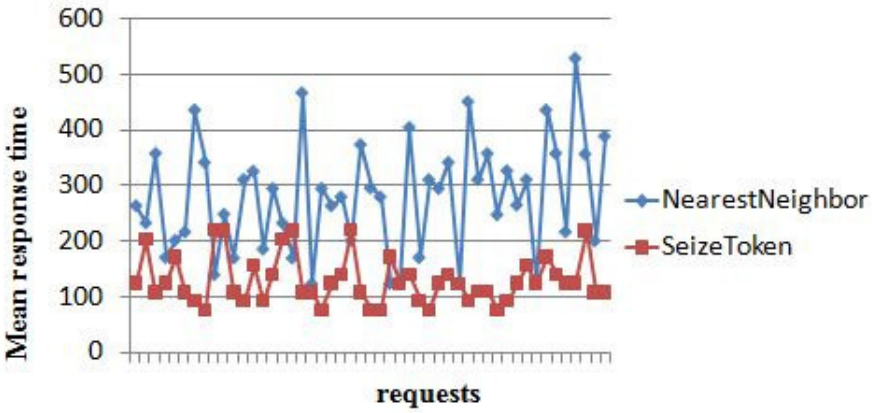

Fig1. Comparison of response times of NearestNeighbor and SeizeToken Algorithm

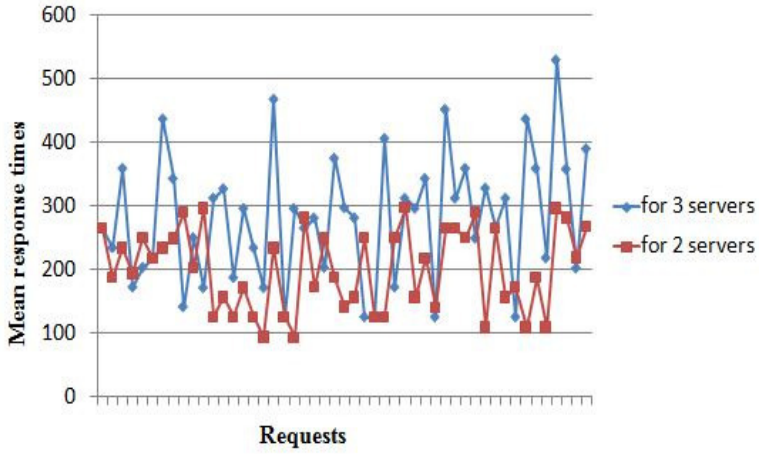

Fig2. Comparison of response times in NearestNeighbor when implemented with 2 servers and 3 servers

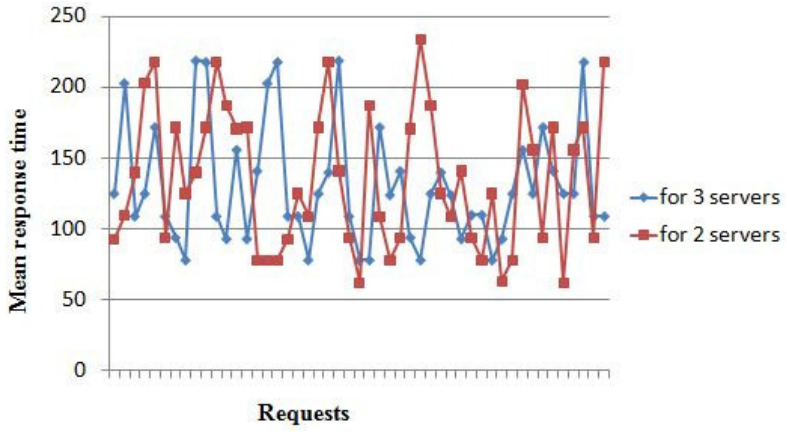

Fig3. Comparison of response times in SeizeToken when implemented with 2 servers and 3 servers

SEEK DIGITAL LIBRARY

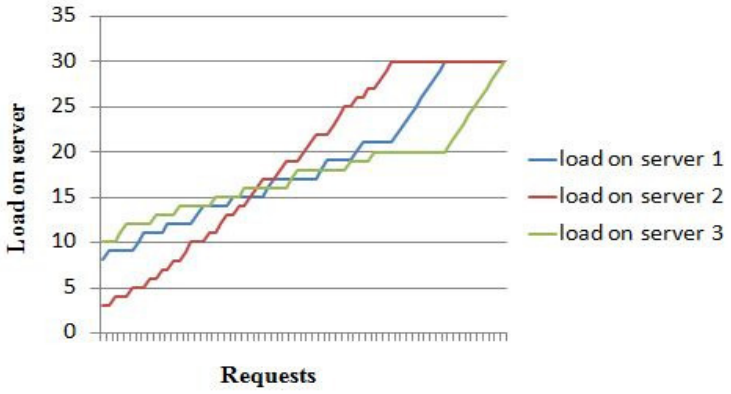

Fig4. Comparison of load on various servers in SeizeToken

VI. Conclusion

In a distributed system, improving dynamic load balancing mainly involved collecting load information from various nodes, determining the least loaded and transferring the task to that node. This obviously keeps each node in the system with minimum possible load and balances the system but this complex strategy sometimes ignores the basic aim of load balancing that is to improve the response time to the client. The overhead involved in transfer of task may lead to degradation of performance. Thus avoiding a completely balanced system solves the problem. At times, a particular given node may have a one or two tasks more than the other but this does not mean it is overloaded and the task should be transferred, it still can execute many more tasks and so should be executed from that node only. This is what came out as a part of our simulation results. Also the technique does not involve any single point of failure. Since each node is operating completely independently, failure of any node does not have any major adverse effect on the system.

REFERENCES

[1] A.M.Alakeel, "A Guide to Dynamic Load Balancing in Distributed Computer Systems."International Journal of Computer Science and Information Security, vol. 10,no. 6,pp 153-160,2010

[2] D. Eager, E. Lazowka, and J.Zahorjan, "Adaptive load sharing in homogeneous distributed systems," IEEE Trans. On Software Engineering, vol. 12, no. 5, pp 662-675, 1986.

[3] S. Shatz, J. Wang, and M.Goto, "Task allocation for maximizing reliability of distributed computer systems," *IEEE Trans. On Computers*, vol. 41, no. 9, pp. 1156-1168, 1992.

[4] A. Elsadek and B.Wells, "A heuristic model for task allocation in heterogeneous distributed computing systems," *The International Journal of Computers and Their Applications*, vol. 6, no. 1,1999.

[5] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Computer. Communication. Rev. 31, 4 (August 2001), 149-160

[6] A. Karimi, F. Zarafshan, A. b. Jantan, A. R. Ramli and M.I. Saripan,"A New Fuzzy Approach for Dynamic Load Balancing Algorithm," International Journal of Computer Science and Information Security," vol. 6 no. 1, pp. 001-005, October 2009.

[7] B. Blake, "Assignment of Independent Tasks to Minimize Completion Time," Software-Practice and Experience, Vol. 22, No. 9,pp 723-734, September 1992.

[8] Mitzenmacher, M., "How useful is old information?," Parallel and Distributed Systems, IEEE Transactions on , vol.11, no.1, pp.6-20, Jan 2000

[9] Tada, H., "Nearest Neighbor Task Allocation for Large-Scale Distributed Systems," Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on ,vol.,no., pp.227-232, 23-27 March 2011.

SEEK DIGITAL LIBRARY