Token Based Load Balancing in Case of Process Thrashing

P.Neelakantan, M.M.Naidu

pneelakantan@rediffmail.com

***Abstract*—**Load balancing is defined as a process of transferring the workload from heavily loaded nodes to the lightly loaded nodes so as to equalize loads at all nodes. But in the process of load balancing there is a possibility that several heavily loaded nodes can transfer their workloads to a same under loaded node which in turn cause it to be overloaded, which again the overloaded node will then transfer some of the workload to another node causing a thrashing effect. In this paper we propose an algorithm which uses a token concept for balancing load among the nodes. Instead of each node probing other nodes often resulting in unsuccessful probing, we take a different approach that makes the identity of the heavily loaded and lightly loaded nodes better known to other nodes**.**

I. INTRODUCTION

The purpose of the load balancing is to achieve some performance goal(s),such as improving mean response time, maximizing resource utilization. Generally a load balancing algorithm has four components: a transfer policy that determines whether a node is in a suitable state to participate in a task transfer, a location policy that determines to which node a task selected for transfer should be sent, a selection policy that determines which task should be transferred, an information policy which is responsible for triggering the collection of system state information. A transfer policy typically requires information on the local node's state to make decisions. A location policy, on the other hand, requires information on the states of the remote nodes to make decisions. There are two different types of algorithms which will initiate the load balancing activity. In sender-initiated algorithms, load balancing is initiated by the overloaded node that attempts to send a task to an under loaded node. Here the overloaded node is the sender and the under loaded is the receiver. But making the overloaded node to send the tasks to the under loaded node which pose an extra workload to the overloaded node which would be a performance bottleneck. In receiver initiated load balancing, the initiating activity is done from under loaded node. In both the algorithms, a threshold value which is based on CPU queue length is used for transfer policy. The selection policy selects a task newly originated tasks that caused the node to become overloaded. The information policy plays an important role in transferring the load from the heavily loaded nodes to lightly loaded nodes. This policy is responsible for deciding when information about the states of other nodes in the system should be collected. The information policies used in distributed load balancing falls under three categories: Demand-driven, periodic and state-change driven. In demand –driven a node collects the state of other nodes only when it becomes either sender or receiver, making a suitable candidate for initialing the load balancing among the nodes. In sender-initiated policies, sender looks for receivers to transfer their load. In receiver- initiated policies, receivers solicit bids from senders. In a periodic policy, nodes exchange load information periodically. But periodic policies do not adapt their activity to the system state. In state-change-driven policy, nodes send their state information whenever their state changes by a certain degree. A state-change driven policy varies from demand-driven policy in that it disseminates information about the state of a node (overloaded or under loaded) rather than collecting the information about other nodes.

In all the above methods, there is a possibility that overloaded nodes can detect the same node with less workload and they will try to send their loads to the destined under loaded node which now becomes overloaded, again tries to transfer the workload to other underloaded node in the given network which results in thrashing effect. Thrashing or instability is a problem that needs to be addressed in any load balancing schemes. In this paper we propose an algorithm which uses a token concept for balancing all nodes. For that it requires the network that is to be organized as a logical ring for an existing network.

II. BACKGROUND &NOTIONS

Consider a network consisting of *n* nodes with *n-1* links connecting all nodes. In this paper we have assumed ring topology. In load balancing schemes first we have to determine whether a node is underloaded or overloaded. To do this we have to evaluate load at each node. To consider load, a comparative study of different load indices carried out by Ferrari et al. reported that CPU load information based upon the CPU queue length does much better in load balancing compared to CPU utilization. The reason CPU queue length did better is probably because, when a host is heavily loaded, its CPU utilization is likely to be close to 100% and it is unable to reflect the exact load level of the utilization. In contrast, CPU queue lengths can directly reflect the amount of load on a CPU.For load balance to be useful, one must first determine when to load balance. The load balance of a computation is the ratio of the average computer load to the maximum computer load,

$$\Gamma = \frac{L_{avg}}{LMax} \quad \text{---------} \rightarrow (1)$$

The load balancing activity is initiated whenever the Γ(load balance) of a computation is below some user specified threshold Γ_{Min}. The L_{avg} is calculated initially by use of discovery token. But the above method is not suited for situations in which the load is changing. In this case when can node is overloaded when $\Gamma > \Gamma_{Min,}$ otherwise underloaded. Another factor that is to be considered in migrating the load from underloaded nodes to the lightly loaded nodes is the communication latency. Latency in a network is measured either *one-way* (the time from the source sending a packet to the destination receiving it), or *round-trip* (the one-way latency from source to destination plus the one-way latency from the destination back to the source). Round-trip latency is more often quoted, because it can be measured from a single point. Note that round trip latency excludes the amount of time that a destination system spends processing the packet. Many software platforms provide a service called ping that can be used to measure round-trip latency. Ping performs no packet processing; it merely sends a response back when it receives a packet (i.e. performs a no-op), thus it is a relatively accurate way of measuring latency. If the cost of load balancing would exceed the benefits of a better work distributed , then it may be better not to load balance. The expected reduction in runtime due to load balancing can be estimated loosely by assuming efficiency which will be increased to $\Gamma_{Min.}$

III. PROPOSED METHOD

In order to get information from other nodes, each node sends a request asking their load. Let us assume that every node collects the information from other nodes in (i.e.,) P_i collects information from other j nodes in the network in the following manner.

P_j	Load $_j$

Where $j \in 1, 2...N$ and $j \notin i$.

After receiving the information from the remaining nodes, each node sorts the loads in non decreasing order to find a node with least load. Here there is a possibility that multiple nodes can choose the same node to transfer the workload to that particular node which has been made overloaded. Then that overloaded node collects the information as said earlier and do the same where the same situation has been repeated. An algorithm is said to be unstable if it could enter a state in which all the systems are spending all of their time migrating processes without accomplishing any useful work in an attempt to properly schedule the process for better performance. This is known as process thrashing. In our algorithm, the above mentioned way of collecting information is avoided which can solve the thrashing effect.

Token:

Here the token can be of two forms. The first form of the token is called discovery token and the second form of the token is called distribution token.

Discovery token:

With the discovery token, the node that has initiated the token can find the total workload of the system and also it finds the number of nodes in the given network.

The format of discovery token is as shown below

SID	SUCCID	LI	INC

Here the SID refers to sender- Identification and SUCCID refers to Successor node, LI is the Load Indexing, INC is the pointer incremented at each node. Initially the value of INC is 1. The token circulates round the ring, and comes to the sender node which has initiated the discovery token. As each node knows the predecessor and successor nodes in the network (which is the principle of logical ring), it knows that it has received correctly from the node that it is expected from. Load index is the field where each node writes its load summed with already the load that is existing in the field. Generally it takes the form

Load index = Previous load index+ Load Index of current node.

The Initial value of the load index is the load of the sender node which has initiated the discovery token, then that value is updated by the next node by adding its load to the load index field.

Issue in Discovery Token: Here the issue is which node has to transmit the discovery token. If we use receiver-initiated approach, then the under loaded node has to initiate the discovery token but in a given network there may be more underloaded nodes, so, there is a possibility of more than one token can circulate around the ring. The same may be applicable to the sender-initiated approach. Here also there is a possibility of more overloaded nodes, which in turn all these nodes initiate discovery token, so more than one discovery token is found in the network. Our objective is to make only one discovery token that is to be circulated around the ring. In order to do this, we assume that a system with largest IP address initiates the activity.

Distribution token: Once the job of the Discovery token is over, the sender node which has initiated Discovery token will covert it into distribution token. The distribution token takes the form

SID	LA	LSN

Here SID is the sender-Identification, LA is the load average of the network and LSN is the Load to be transferred to the successor node.

The node sends nothing if its load is less than the average workload. But if the node is having the load greater the average

workload, then it sends the portion of the workload LSN= $\overline{w} - w_i$, where $\overline{w}$ represents the average work load and w_i represent the workload of node i to the successor node. In this way the token is circulated rround the ring and it reaches the origin (the node which has initiated the token). By now we can tell all the nodes in the ring have the same work load, $w_i = \overline{w}$ .But one thing that is to be remembered here is always the node sends its excess load to its successor not to the predecessor. We also assume that no workload is generated during the token circulation in the ring. That is, as the load balancing progresses, every node should be instructed to take or to give certain amounts of workload with respect to each of it's logically directly connected successors.

IV. ALGORITHM

Algorithm Balance ()

{
// choose a machine with highest IP address. Then build Discovery Token which is to be circulated round the ring.

1. *Send Discovery token to the successor. Initially load Index field contains the load of the first processor (i.e.,) a machine with highest IP address. INC is initialized to 0.*

// All systems in the network will update their loads in the load index field and INC is incremented by one when it encounter a new node in the network. Here 1 refers to the machine with largest IP address and N-1 refers to predecessor node to 1. As 1 knows that its predecessor is n-1 in the logical ring, it knows that the token is circulated round the ring.

2. *Load Index=0 ;*
   INC=1;
   For i= 1 to N-1 MOD N
   Load Index = Load Index + Load of Index (i);

// The initial sender will calculate the average load from the information that has been received from token .

3. *Average Load=* $\frac{loadindex}{INC}$ *;*

   // the initial sender which has initiated the discovery token which change the token format into Distribution token format.

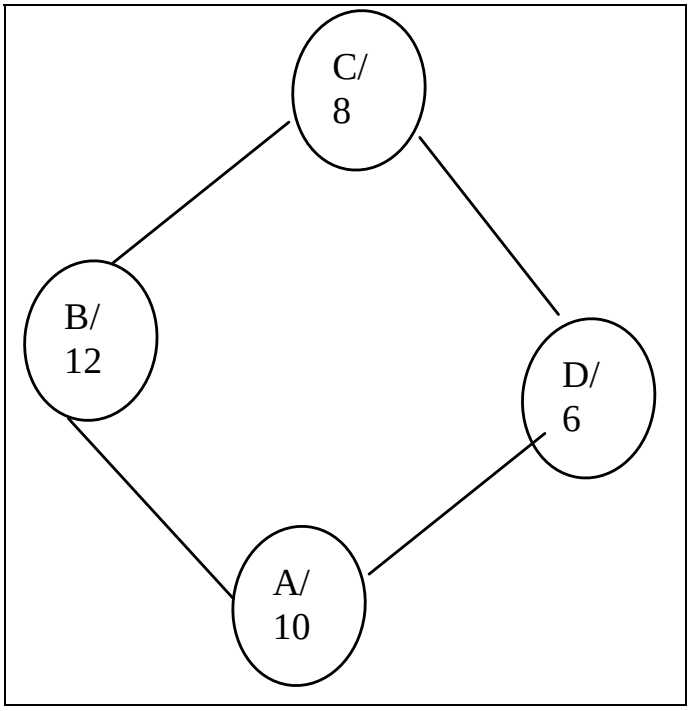
4. *LSN=0*
   for i = 1 to N-1 MOD N
   a. LA= Average Load;
   if (Li> LA)
   a. LSN=(L_i –LA)+LSN;
   b. Li= Li- LA

   else

   // create temp variable LSN1 in a node that is holding token
   a. LSN1= LSN;
   b. LSN= LSN- (LA- L_i);
   // transfer (LSN1- LSN) amounts of work load to the node i and update the workload
   // of node i
   c. Li= Li+ (LSN1- LSN);

// End of for loop
} // End algorithm Balance

V. EXAMPLE



As per our algorithm proposed in later section, we will take the above graph to illustrate the proposed algorithm. As shown above there are four vertices which represent nodes. Each vertex represents the name of the node with load index. We will assume that Node A is having the highest IP address which will initiate the load balancing activity by calling our algorithm balance. As per our algorithm first it sends a discovery token by setting load index into its load value (i.e.,) load index =10 and Inc =1 to its successor B, which updates load index=22 and inc=2 after last iteration the load index =36 and Inc=4 and the discovery token that is containing this information is received by A which it knows that it receives the token from D. After receiving the load index is divided by the Inc which will yield the average load. Now the Node A can convert the discovery token to distribution token which is to be transmitted to the node B. The distribution token contains Load average value=9 and LSN=1. But at node B, *if* statement of our algorithm is executed so, no workload is assigned to B, because the workload of B is higher than the average load. So, the else statement is executed and the LSN field is updated with the

excess load of B and hence LSN=4. At Node C, the *else* statement is executed and (LA-$L_{i)}$ amounts of workload is and the workload of node C is updated and LSN has been reduced by (LA- L_i) $_{units}$. At node D, once again the else statement is executed and (LA-L_i) amount of workload has been transferred from LSN field.

According to our Distribution token, the following information will be available in token circulation round the ring.

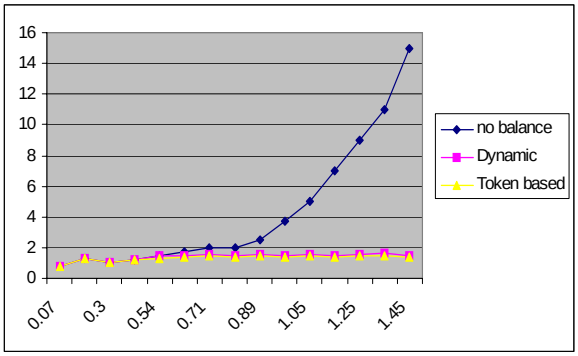
Now by using the above example network we have proved that the algorithm works well and it has done perfect load balancing

SENDER –ID	LOAD AVG	LOAD TO SUCCESSOR NODE	Action
SID=A	LA=9	LSN= 1	-
SID=B	LA=9	LSN=4	-
SID=C	LA=9	LSN=3	It takes one load unit.
SID=D	LA=9	LSN=0	It takes three load units.

in all the nodes in a given ring topology.

VI. SIMULATION

We have taken the parameters from [9] and here we assumed that the tasks of fixed size with an average service time (exponentially distributed) equals to 1 with respect to a reference processor .For all simulated models, the parameters of the simulated models are the following: Number of processors=16, token transfer delay=0.01, task transfer overhead=0.1, message transfer overhead=0.01, task average service time=1, Threshold=3, probing communication overhead=0.02, probing limit=1. Token transfer delay is chosen such that the average response time does not change significantly when smaller value is used. Message transfer overhead is the time required to send a message to next node. The probing limit and threshold values were taken from [10].



REFERENCES

[1] D.Karger, M.Ruhl. "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems". *In Proc. SPAA*, 2004.

[2] P Brighten Godfrey, Ion Stoica. "Heterogeneity and load balance in Distributed hash tables". *In Proc. IEEE INFOCOM*, 2005.

[3] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp,and I. Stoica. Load balancing in structured p2p systems. In *Proc. of IPTPS*, 2003.

[4] H. Shen and C. Xu. Locality-aware randomized load balancing algorithms for structured p2p networks. In *Proc* Ananth Rao, Kartbik Lakshminarayanan, Sonesh Surana, et al. "Load Balancing in Structured P2P Systems". *In Proc. IPTPS*, Feb, 2003.

[5] P Brighten Godfrey, Karthik Lakshminarayanan, et al. "Load balancing in dynamic structured P2P systems". *In Proc. IEEE INFOCOM*, 2004.

[6] Ganesan P, Bawam. "Distributed balanced tables: not making a hash of it all". Stanford University, Database Group, 2003.

[7] H. Shen and C. Xu. Locality-aware randomized load balancing algorithms for structured p2p networks. In *Proc. of ICPP*, pages 529–536, 2005.

[8] Mukesh Singhal, Niranjan G. Shivaratri,"*Advanced Concepts in Operating Systems*", McGraw-Hill; ISBN: 007057572X.

[9] Tariq Omari, Seyed H.Hosseni, K.Vairavan, " Travelling token for Dynamic Load Balancing", Proceedings of the Third IEEE International Symposium on Network Computing and Applications(NCA'04).

[10] O.Kremien,J.Kramer, " Methodical analysis of adaptive load sharing algorithms", IEEE trans.on parallel and distributed Systems, vol.3, No.6, 1992.

[11] Plamenka Borovska, Milena Lazarova, "Token – Based Adaptive Load Balancing for Dynamically Parallel Computations on Multicomputer Platforms", International Conference on Computer Systems and Technologies, CompSysTech, 2007.

[12] Jerrel Watts and Stephen Taylor," A Practical Approach to Dynamic Load Balancing", IEEE trans.on parallel and distributed systems, Vol.9, No.3, March 1998.

[13] Bertsekas, D. P. and Tsitsiklis, J. N. (1997), *Parallel and DistributedComputation Numerical Methods*, Athena Scientific, Belmont.