A Dynamic Sliding Load Balancing Strategy in Distributed Systems

Ahmad Dalal'ah
Computer Science Department, Jordan University of Science and Technology, Jordan

Abstract: *A sliding strategy for load balancing is introduced. The strategy groups a certain number of adjacent nodes to perform a load balancing process. Upon the completion of a given period, the groups are to be rotated by shifting each group one position to the right, thus produces different groups. This strategy (sort of clustering) not only reduces the load balancing overheads, but also it could be utilized as a backbone by any load balancing strategy. The proposed load balancing strategy always converges, and tends to be in a steady state in a negligible processing time. In this paper, the load status and the locations of the nodes regarding the system's topology are irrelevant to load balancing process. The new algorithm can be always applied to any distributed system, even if it is heavily loaded, since the cost of scheduling is very low due to the highly reduced number of messages. This is achieved by reducing dramatically the overheads incurred from attached information tables, message passing, job thrashing, and response time.*

Keywords: *Load balancing, distributed systems, scalability, message passing.*

Received January 10, 2005; accepted March 20, 2005

1. Introduction

The growth of distributing systems with the possibility of sharing the available resources has led to the ability of executing some jobs arrived at a certain node remotely. Consequently, load balancing is aiming at reducing the overall response time of jobs execution in any distributed system. This increases the possibility of maximizing the overall utilization of any given distributed system by sending some of the tasks at highly loaded nodes to be executed remotely. Such process can be profitable if the gain obtained from load balancing outperforms the execution of all the jobs locally, in terms of the number of executed tasks in time unit, and the number of accepted tasks if the queues are of limited length.

Almost all scheduling strategies in distributed systems depend on load status and locations of the system nodes. The used policies are normally classified under centralized, distributed and internally source or server initiative and others. Such classification entails unavoidable overheads which stem from the fact that heavy computations are necessary to balance the load. These heavy computations are subject to be minimized.

To take a reasonable decision of sending a job to be executed remotely or not depends, to some degree, on the amount of information available at the load balancer node. At such node a formal assessment is needed to be done to take a right decision of where to perform the newly arriving job. The information needed to take a decision by a load balancer is composed of the status of each other node in the system, the location of the nodes that are willing to accept the transferred jobs, and some other information depending on the load balancing strategy applied, such as the criteria of transferring a job.

As long as an abstract structure of a certain node is concerned, the node can be visualized as a queue and a processing unit. As a consequence, every arriving task is to be queued waiting for execution if the job's arrival rate is more than the job's served rate. The round response time could be expressed as the time spent by a task since its arrival up to the end of its execution time.

As mentioned above, the information needed at each node includes the location of the peer node that is willing to accept a task or even to transfer a job. This information should be on-line. Otherwise, the delayed information may lead to a situation where a node that appeared lightly loaded is no more as it was, and hence, all the transferred jobs are to suffer from waiting to be transferred again to a different node (job thrashing). On the other hand, the process of locating the peer node(s) may involve a high number of messages to be communicated among different nodes in the system. This might be an overhead that causes a delayed response time.

Having located a peer node; the issue of either processing the task locally or remotely is to be triggered. Given the information available at each node, the decision of transferring a job is to be taken. Hence, the time necessary to transfer a job and to receive its results is to be taken into account. Hereby, another type of overhead is incurred. This overhead depends not only on the size of the task to be transferred, but also on the distance between the

partner nodes, although this last problem might be bypassed due to the present high-speed communications facilities available.

A great deal of effort has been devoted to the scheduling problem. Many of these efforts can be found in the literature. In [11] the authors tried to solve this issue as a nonlinear programming problem. Their measure focused on the average job response time on a probabilistic basis. This metric is very important to evaluation, but still, other metrics such as the number of messages passed through the load balancing process are rather important and should be taken into account as well. The authors in [5] tried to control the overheads in adaptive load sharing, in one of their proposed algorithms; they used a heuristic scheme to guess the load level at different nodes, while in their second algorithm they probe the jobs and decide not to transfer the unnecessary ones. However, in the first algorithm it is not guaranteed to know the load level in advance, while the other algorithm is biased against some jobs and involves extra computation to decide which of the jobs is to be transferred and which should not. P.Krueger and *et al.* in [6] proposed an adaptive strategy to probabilistically select the best counterpart of a generic node, if it exists to transfer some of its load to be executed remotely. The adaptation process involves extra computations and remains location dependent. Others as in [6] used a sender-initiated algorithm to solve the load balancing problem; this class of algorithms may suffer from instability and do not using the maximum capacity of the system. Efe *et al.* in [5] proposed a central job dispatcher for load balancing. It is true that the centralizing nature of load balancing can, under some constraints, give better response time than other strategies. However, they do not only suffer if the center has problems, but also their decisions could be valueless due to the delayed information used by the decision maker.

The proposed approach in this paper takes into consideration the number of messages exchanged. To this aim, the number of exchanged messages is fixed in each load balancing period regardless of the system size (when the system consists of groups of three nodes), while it was 2 (n - 1) in other works [2, 12, 13], where n is the number of nodes. This would be an evident enhancement on the overall system throughput. Another issue is the location problem. This is totally avoided due to devising a controller node in each group that acts as a local load balancer. Therefore, the domain of the controller is limited to its direct neighbors.

The rest of the paper is organized as follows. In section 2 the proposed strategy is described. Numerical results are in section 3. Conclusions are shown in section 4.

2. Proposed Strategy

The overheads normally incurred from implementing any load balancing policy are always subject to strategies aiming at reducing such overheads. Here, in this work, we tried to reduce the overheads stem from the communication problem (message passing and relatively long distance transfer), location problem or constructing clusters, and job thrashing. Many other problems are faced and to be discussed later, such as stability, scalability, robustness, and efficiency.

2.1. System Topology

The system is supposed to be some sort of network, connected as a closed bus and the nodes are to be logically ordered from 1 to n, where n is the total number of nodes in the entire system as shown in Figure 1. In this system, two methods were used to configure the topology. In the first method, the nodes are coupled in pairs, peer-to-peer, where no contentions suppose to happen; i. e., each couple of nodes stands alone and it is not in touch with any other node at any moment for the purpose of load balancing. However, these couples are to be changed at every load balancing process, as it is addressed here after in this subsection. The cost of changing the groups is the minimum due to the number of messages that are to be exchanged. While in the second method, the group of nodes that share the load consists of three nodes each, the process of grouping is discussed later in this section as well.

Since the system is tested over 2 and 3 nodes groups. Figure 2 shows the topology of a sample network before and after the first load balancing operation. Figures 2-a and 2-b show the grouping policy applied when the system is coupled in pairs. While Figures 2-c and 2-d show the connected nodes as groups of threes. Figures 2-b and 2-d reveal the connected nodes after a load balancing process.

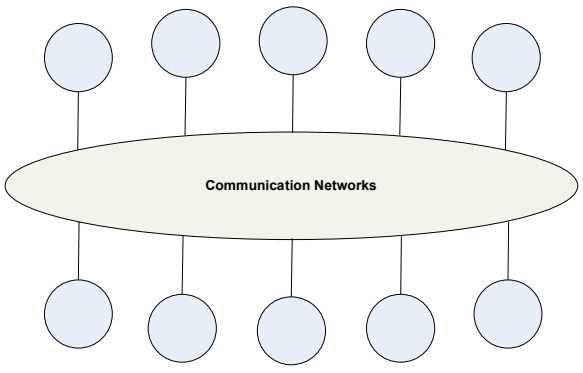

Figure 1. System topology.

To select the nodes of each group in a dynamic manner, we applied the pseudo-code shown in Figure 3. Each group could be constructed of any number of nodes. But it is noticed that when the groups' size exceeds 3, then the entire system will be changed into

a pure clustering system, and in this case it would not be different from any other clustering strategy. Therefore, if the number of nodes becomes more than 3, then the system might loose the gain that may be obtained from the proposed strategy in terms of control messages number, growth of transferring distance, and its dynamic nature.

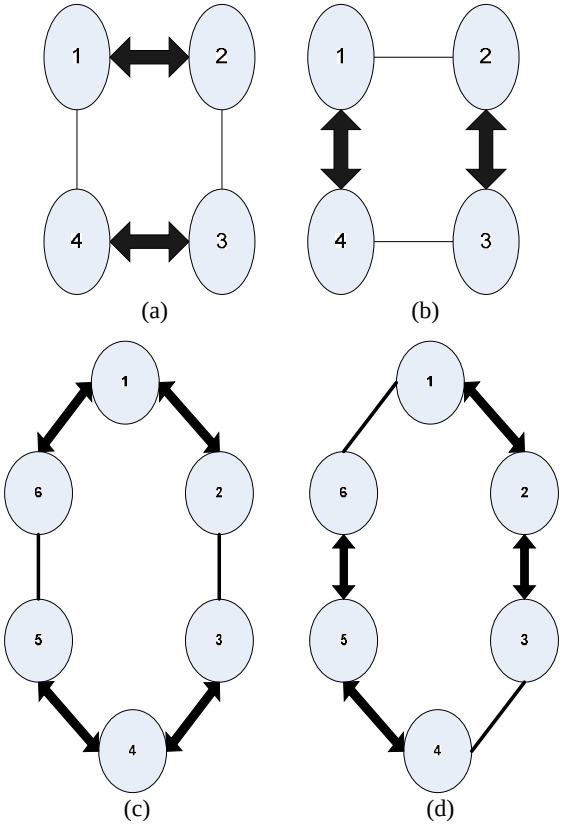

Figure 2. Coupling process after each load balancing operation.

Since the system groups the nodes into threes, then the system would scale and the size of the network would be irrelevant. On the other hand, job thrashing is avoided here; i. e., it would not happen that many nodes would dump many jobs over a lightly loaded node at a given period. Otherwise, the tasks are to be swinging from one part to another on the network, and the system would be instable.

Another issue is the robustness; the proposed strategy is robust to any failure that may occur. This is achieved simply because if any node failed then only at that moment no load balancing happens in that particular group. At any other moment the topology is changed, the failure node is ignored and the system carries on without any problems.

The following mechanism is used to change the coupled nodes in a dynamic manner. Upon each load balancing process, the pseudo-code in Figure 3 is triggered and each couple of nodes is to be changed. This gives the proposed policy the potential of reducing the overheads mentioned above in terms of message passing and selecting a coordinator. For example, if we take the topology in Figure 2-c and 2-d then the grouping mechanism would comply with the following pseudo-code:

```
n := NetworkSize;
k := 0;
l := 0;
j :=1;
for i := j + 1, n + 1, STEP 3;
    begin
        k := k + 1;
        C (k, 1) := i - 1;
        C (k, 2) := i;
        If I + 1 >= n then C (k, 3) :=(I + 3) mod n
          Else
            C (k + 3) :=I + 1
    end
```

Figure 3. Pseudo-code for constructing neighboring groups.

2.2. Load Balancing Strategy

Two conditions are necessary to activate the load balancing process, the first happens when the local load exceeds a predefined threshold. At this stage the coordinator of each group communicates with its neighbors and distributes the load almost evenly among the related nodes. The second case of load balancing is performed periodically. Hereby, load balancing is to be performed only if the load at the coordinator node is still over the previously mentioned threshold.

The policy is, first, applied over each group of nodes. In this case, if nodes *I*, *J* and *K* are grouped then the load over such nodes will become:

$$Netload = int((load\ i + load\ j) / 2), \text{ for 2 nodes groups (1)}$$

$$Netload = int((load\ i + load\ j + load\ k) / 3, \text{ for 3 nodes groups (2)}$$

Where, Netload is the number of jobs on each node after load balancing, load *i* is the number of jobs on node *i*.

This process is done simultaneously at each couple of nodes. This could be achieved by applying the linear algorithm mentioned in Figure 3, but instead of STEP = 3, it becomes equal 2.

On the other hand the proposed strategy gets red of location problem, in which the node should search for its partner in the load balancing process.

3. Numerical Results

The scalability of the proposed system highlights the fact that the overall cost relative to the size of the entire system is irrelevant. This result stems from the fact that the system is to be viewed only by 2 or 3 nodes depending on the method used that makes the cost almost fixed. While in other policies [1, 3, 4, 7, 8, 10, 12, 13, 14, 15, 17, 20, 21] the overall cost incurred from the load balancing process is size dependent and hence it would not scale.

In the proposed algorithm, as mentioned above, the number of messages is fixed (in case of 2 nodes groups, the number of messages would be at most 2, while in 3 nodes groups, the number of messages would be at most 5). On the other hand, in Central algorithms the number of messages is 2 (n - 1) and in Distributed algorithms the number of messages is (n - 1) / 2, where n is the number of node in the entire system.

One more advantage of our algorithm is that the nodes involved in the load balancing process are only the groups that are really in need for load balancing, while in other policies the entire system would be involved. In fact, only in an extreme case, all the groups are to be balanced. This reduces the overall cost on average.

Figure 4 shows the effectiveness of the proposed strategy, in both cases 2 and 3 groups. In this sense, the overhead incurs from messages exchanged among different nodes is no longer relevant, since it is fixed and very low.

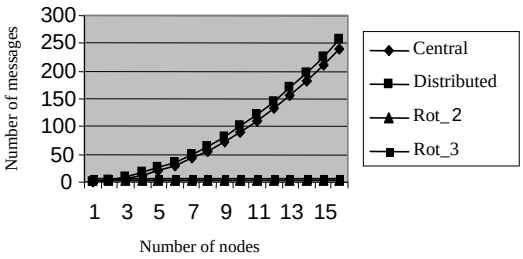

Figure 4. Number of messages exchanged for central, distributed, and the proposed two policies.

One more advantage issue is the number of success jobs relevant to the overall jobs in the system. It is clear that our strategy outperforms other strategies due to the fact that the percentage of the tasks that are prone to be transferred from one node to another without being executed is very slim. Hence, the utilization of the proposed system outperforms the other cases in the conducted experiment. While in other policies such possibility is very high [7, 14]. For example, it might happen that a certain node announces itself as a lightly loaded node, and as thus many other nodes will transfer their surplus load to it, which might cause the node to be an extremely highly loaded one. As so, such node starts hunting another node to transfer extra tasks to it. Another point in our strategy worth noting is the possibility of accepting a transferred job or more while many other jobs arriving at it is very low. This is because load balancing process happens to be among adjacent nodes only. Figure 5 shows the reaction of our system in comparison to the no load balancing and to a clustering load balancing strategy [1].

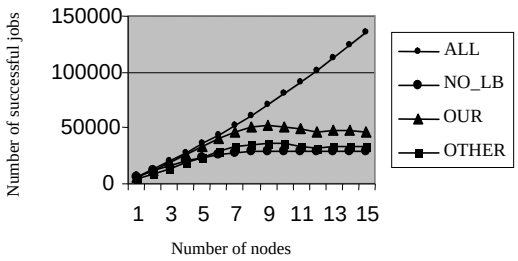

Figure 5. Number of success jobs relative to the total number of jobs for the three compared policies.

4. Conclusions

The overheads stem from message passing and the scalability issues are among the main objectives of any load balancing system. To this aim, two methods of grouping the nodes were introduced, devised and tested. The first is to group the nodes in couples while the second one is to group the nodes into triples. The numerical results show that the overhead stem from computations is reduced dramatically in both methods. On the other hand, the number of messages is not any more an important issue, since it turns to be fixed with small number of messages as well as the utilization of the system is maximized. The proposed policies guaranteed the distributed system to be scalable.

References

[1] Chen M. S. and Shin K. G., “Sub Cube Allocation and Task Migration in Hypercube Multiprocessor,” *IEEE Transactions on Computers*, vol. 39, no. 9, September 1990.

[2] Dalal’ah A. and Al-Dahoud A., “Dynamic Load Balancing with Token-Passing in a Distributed System,” *IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, California, USA, pp. 398-400, October 1998.

[3] Eager D. L., Lazowska E. D., and Zahorjan J., “Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Transactions on Software Engineering*, vol. SE-12, no. 5, May 1986.

[4] Eager D. L., Lazowska E. D., and Zahorjan J., “The Limited Performance Benefits of Migration Active Processes for Load Sharing,” *in Proceedings of ACM Sigmetrics Conference*, pp. 662-675, 1988.

[5] Efe K. and Groselj B., “Minimizing Control Overheads in Adaptive Load Sharing,” *The Center for Advanced Computer Studies*, University of Southwestern Louisiana, Lafayette, USA, 1989.

[6] Hac A., “Dynamic Load Balancing in a Distributed System Using a Sender-Initiated Algorithm,” *Journal of System Software*, vol. 11, no. 2, pp. 79-94, 1990.

[7] Hagmann R. B., "*Process Server*," *in Proceedings of the 8th International Conference of Distributed Computing Systems*, Cambridge, Mass, pp. 260-267, May 1986.

[8] Kara M., "Using Dynamic Load Balancing in Distributed Information Systems," *Report 94.18*, School of Computer Studies, University of Leeds, May 1994.

[9] Krueger P. and Shivaratri N. G., "Adaptive Location Policies for Global Scheduling," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, June 1994.

[10] Leland W. and Ott T., "Load Balancing Heuristics and Process Behavior," *in Proceedings of ACM Sigmetrics Conference*, pp. 54-69, May, 1988.

[11] Ni L. M. and Hwang K., "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 5, May 1985.

[12] Ni L. M., Xu C., and Gendreau T. B., "A Distributed Drafting Algorithm for Load Balancing," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 10, October 1985.

[13] Pinter S. S. and Woltstahl Y., "On Mapping Processes to Processor in Distributed Systems," *International Journal of Parallel Programming*, vol. 16, no. 1, 1987.

[14] Rao G. S., Stone H. S., and Hu T. C., "Assignment of Tasks in a Distributed Processor System with Limited Memory," *IEEE Transactions on Computers*, vol. C-28, no. 4, April 1979.

[15] Shin K. G. and Chang Y., "Load Sharing in Distributed Real Time Systems with State-Change Broadcasts," *IEEE Transactions on Computers*, vol. 38, no. 8, August 1989.

[16] Tantawi A. N. and Towsley D., "Optimal Static Load Balancing in Distributed Computer Systems," *Journal of the ACM*, vol. 32, no. 2, pp. 445-465, April 1985.

[17] Xu J. and Hwang K., "Heuristic Methods for Dynamic Load Balancing in a Message-Passing Multicomputer," *Journal of Parallel and Distributed Computing*, vol. 18, no. 1, pp. 1-13, 1993.

[18] Zein O., El-Toweisy M., and Mukkamala R., "A Distributed Scheduling Algorithm for Heterogeneous Real-Time Systems," *in Proceedings of Advances in Computing and Information (ICCA'91)*, Ottawa, Canada, pp. 27-29, May 1991.

[19] Zhou S., "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1327-1341, September 1988.

[20] Zhou S. and Ferrari D., "An Empirical Investigation of Load Indices for Load Balancing Applications," *in Proceedings of the International Symposium on Computer Performance Modeling Measurement and Evaluation (Performance'87)*, Brussels, Belgium, pp. 515-528, 1987.

[21] Zhou S. and Ferrari D., "An Experimental Study of Load Balancing Performance," *Technical Report UCB/CSD 87/336*, University of California, USA, January 1987.

Ahmad Dalal'ah received his BSc in computer science from Yarmouk University, Jordan in 1985. During the period 1989-1993 he was a TA at Mu'tah University, Jordan. He received his PhD in computer networks in 1998 from Genoa University, Italy. Currently, he is with the Computer Science Department at Jordan University of Science and Technology. His research interests include ad hoc networks and load balancing in distributed systems.