Load Balancing Algorithms for Peer to Peer and Client Server Distributed Environments

Sameena Naaz Afshar Alam Ranjit Biswas

Department of Computer Science Jamia Hamdard, New Delhi, India

ABSTRACT
Advancements in hardware as well as software technologies have resulted in very heavy use of distributed systems. Because these systems are physically separated so managing the various resources is one of the challenging areas. In this paper will talk about the management of the processing power of the various nodes which are geographically apart. The basic aim is to distribute the processes among the processing units so that the execution time and communication delays can be minimized and resource utilization can be maximized. This distribution of processes is known as load balancing. Load balancing can either be static or dynamic in nature. It has been proved that dynamic load balancing algorithms give better result as compared to static algorithms but they are computationally more intensive. This paper compares the various load balancing algorithms quantitatively. An important issue with dynamic algorithms is that they exchange state information at frequent interval to make decisions. Because there is some communication delay also before the information reaches its destination so there is some uncertainty in the global state of the system. To overcome this problem fuzzy logic concept can be used which is also discussed here.

General Terms
Load Balancing Algorithms

Keywords
Distributed Systems, load balancing, execution time, resource utilization, uncertainty, fuzzy logic.

1. INTRODUCTION
The fast growth of distributed systems has been possible because of advancements in both hardware as well as software technology. Academicians as well as commercial applications depend heavily on these distributed systems. Managing the resources of these distributed systems is a difficult task because they are separated geographically. One of these resources is the processing power of the various nodes which are far apart. The basic aim in these systems is reduce the mean response time. This can be achieved by distributing the processes among the processing units in a proper manner. This is known as load balancing. The design of these algorithms depends upon several criteria. Some of them are the underlying network topology, communication network bandwidth and job arrival rates. The load balancing algorithms are classified as static and dynamic. In static algorithm [1, 2] there are some factors which are predefined on which the decision depends. The current state of the system is immaterial and hence these algorithms are not suitable for real time applications. On the other hand we have dynamic algorithms in which the decisions are taken based on the current state of the system [3, 4 and 5]. In these algorithms the state information is gathered at frequent intervals and used to make decisions. This information exchange takes place in the form of messages which are exchanged. The communication delays in these exchanges result in uncertainty in the state information which is used to make decisions. In order to overcome this problem fuzzy logic concept is used in [6].

2. LOAD BALANCING STRATEGIES
There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. These three parameters answer three important questions:

- Who makes the load balancing decision?
- What information is used to make the load balancing decision?
- Where is the load balancing decision made?

2.1 Centralized Vs. Distributed
In centralized load balancing strategy the responsibility of making the decision and initiating the load balance process lies with a single processor. There is one central processor which gathers the global state information and assigns the various tasks to the processors. These algorithms achieve high performance but suffer from the obvious drawbacks which are associated with any centralized approach. These disadvantages are: high vulnerability to failures, storage requirements for maintaining the state information - especially for large systems, and the dependability of the performance of the system on the central processor which could result in a bottleneck [7].

In a distributed load balancing strategy, same algorithm is executed on all the processors. There is exchange of state information among various nodes and based on the global state information a decision is taken. Each processor may send or receive work on the basis of a sender-initiated or a receiver-initiated policy. Some of the advantages offered by the distributed policy are: Fault tolerance, minimum storage requirements to keep status information, and the availability of system state information at all nodes. But because the system state changes very frequently due to arrival and departure of processes so maintaining and distributing this information is computationally intensive.

To overcome this limitation, some distributed strategies minimize the amount of information exchanged, which has a negative reflection on the performance of an algorithm. The semi-distributed policy comes in the middle between centralized and distributed policies. It is introduced to take the best of each and to avoid the major drawbacks of each of the two policies. The semi-distributed strategy is based on the partitioning of the processors into equal sized sets. Each set adopts a centralized policy where a central processor takes charge of load balancing within its set. The sets together adopt a distributed policy where each central processor of each set exchanges information with other central processors of other sets to achieve a global load balance. It has been shown in [7] that the semi-distributed policy produces a better performance than the centralized and distributed policies. Research demonstrates that each central

processor yields optimal load balance locally within its set. Moreover, this policy does not incur high communication overhead while gathering system state information. Although this policy is a mediator between the centralized and the distributed ones, it fits large distributed systems better than small systems.

2.2 Sender Initiated Vs. Receiver Initiated

In a sender-initiated policy, the sender decides which job gets sent to which receiver. In a receiver-initiated policy, the receiver searches for more work to do. Intuitively, queues are formed at senders if a receiver- initiative policy is used, while they are formed at receivers if a sender-initiative policy is used. Additionally, scheduling decisions are made when a new job arrives at the sender in a sender-initiative, while they are made at the departure of a job in a receiver-initiative policy. The determination of which policy is adopted depends upon the load transfer request which can be initiated by an over-loaded or under-loaded processor [8]. It has been demonstrated in [9] using analytical models and simulations that sender-initiated strategies generally perform better at lower system loads while receiver-initiated strategies perform better at higher system loads, assuming that process migration cost under the two strategies is comparable.

2.3 Global vs. Local Strategies

Global or local policies answer the question of what information will be used to make a load balancing decision in global policies, the load balancer uses the performance profiles of all available workstations. In local policies workstations are partitioned into different groups. The benefit in a local scheme is that performance profile information is only exchanged within the group. The choice of a global or local policy depends on the behavior an application will exhibit. For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time.

3. LOAD BALANCING ALGORITHMS

Load Balancing Algorithms can be classified as static and dynamic

3.1 Static Load-Balancing

In this method, the performance of the nodes is determined at the beginning of execution. Then depending upon their performance the workload is distributed in the start by the master node. The slave processors calculate their allocated work and submit their result to the master. A task is always executed on the node to which it is assigned that is static load balancing methods are non-preemptive. A general disadvantage of all static schemes is that the final selection of a host for process allocation is made when the process is created and cannot be changed during process execution to make changes in the system load [10]**.** Major static load balancing algorithms are Round Robin and Randomized Algorithms, Central Manager Algorithm and Threshold Algorithm.

3.2 Dynamic Load-Balancing

It differs from static algorithms in that the workload is distributed among the nodes at runtime. The master assigns new processes to the slaves based on the new information collected. Unlike static algorithms, dynamic algorithms allocate processes dynamically when one of the processors becomes under loaded. Instead, they are buffered in the queue on the main host and allocated dynamically upon requests from remote hosts. This method is consisted of Central Queue Algorithm and Local Queue Algorithm. Load balancing algorithms work on the principle that in which situation workload is assigned, during compile time or at runtime. Comparison shows that static load balancing algorithms are more stable compare to dynamic. It is also easy to predict the behaviour of static, but at the same time, dynamic distributed algorithms are always considered better than static algorithms [10]

The basic model of a node for load balancing algorithms consists of a scheduler, an infinite buffer to hold the jobs and a processor. The aim of the scheduler is to schedule the jobs arriving at the node so that the mean response time of the jobs is minimized. In the absence of a scheduler in a node, the job flow takes the following sequence of actions: a job enters the buffer, waits in the queue for processing, leaves the queue and gets processed in the processor and then leaves the node (system). However, when a scheduler is present, depending on where a scheduler resides in a node to exercise its control on the jobs, we classify the dynamic load balancing algorithms into three policies

- Queue Adjustment Policy (QAP)
- Rate Adjustment Policy (RAP)
- Combination of Queue and Rate Adjustment Policy (QRAP).

Queue Adjustment Policy (QAP)

In this policy the scheduler is placed immediately after the queue. In these algorithms the aim is to attempt to balance the jobs in the queues of the nodes [3]. When a job arrives at node i, if the queue is empty, the job will be sent to processor directly. Otherwise, the job will have to wait in the queue. The scheduler of node i periodically detects the queue lengths of other nodes that node i can communicate with. When an imbalance exists (some queues are too long and some are too short), the scheduler will decide how many jobs in the queue should be transferred and where each of the jobs should be sent to.

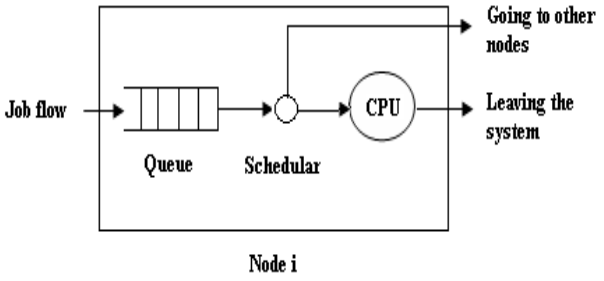

Figure 1(a) Node model of queue adjustment policy

By queue adjustment, the algorithms could balance the load in the system. This scheme is shown in fig 1(a)

Rate Adjustment Policy (RAP)

As shown in Fig 1(b), in this policy the scheduler is placed immediately before the queue. When a job arrives at node *i*, the scheduler decides where the job should be sent, whether it is to be sent to the queue of node *i* or to other nodes under consideration. Once the job has entered the queue, it will be processed by processor and will not be transferred to other nodes. Using this policy, the static algorithms can attempt to control the job processing rate on each node in the system and eventually obtain an optimal solution for load balancing [1]. Because of the high computation overheads, none dynamic algorithm in the literature uses this policy.

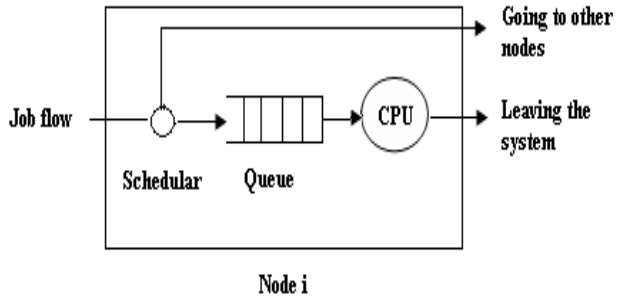

Figure 1(b) Node model of rate adjustment policy

Hybrid Policy: Combination of Queue and Rate Adjustment Policy (QRAP)

In this policy the scheduler is allowed to adjust the incoming job rate and also allowed to adjust the queue size of node i in some situations. Because we consider a dynamic situation, especially when we use RAP, in some cases, the queue size may exceed a predefined threshold and load imbalance may result. Once this happens, QAP starts to work and guarantees that the jobs in the queues are balanced in the entire system. In this policy, we can consider the rate adjustment as a "coarse" adjustment and the queue adjustment as "fine" adjustment. An algorithm based on this policy can be found in [11]. This scheme has been depicted in Fig 1(c).

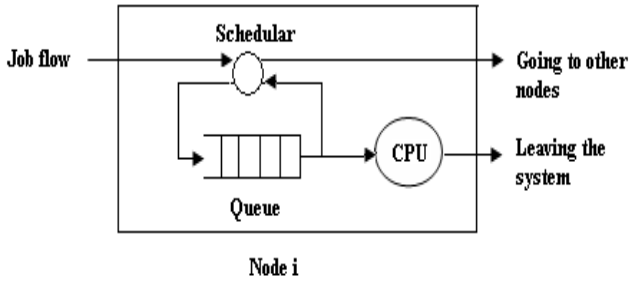

Figure 1(c) Node model of combination of queue and rate adjustment policy.

3.3 Fuzzy Logic Based Load Balancing

There are many load balancing algorithms in literature which reflect state uncertainty in distributed systems. These uncertainties are there because information exchange between various nodes suffers from some communication delays.

The semi-distributed model for task scheduling suggested by [12] can be considered as a trade-off between a centralized and distributed scheduling mechanism. The semi-distributed model is motivated by the fact that uncertainty and message overheads may be reduced by adding the notion of central control to distributed scheduling mechanisms [13] described a fuzzy expert system for load balancing. Each node of a distributed computing system has an expert system that plays the role of a distributed decision maker. The fuzzy expert system reflects the imprecision in state information and makes scheduling decisions based on a fuzzy logic. Reference [14] considered a prediction-based load sharing mechanism. It was argued that the states of remote nodes were better estimated by the prediction of resource requirements for incoming processes. The effect of uncertainty in the decision making process has been studied in [15] which states that the performance of any distributed computing system cannot be improved beyond a limit which is determined by the degree of uncertainty in global state.

4. LOAD BALANCING POLICIES

There are four policy mechanisms on which various load balancing algorithms are based. They are

- Selection Policy
- Transfer Policy
- Information Policy
- Location Policy.

Specifically, information and location policies have the most important roles.

4.1 Transfer policy

First of all the state of the different machines is determined by calculating it's workload. A transfer policy determines whether a machine is in a suitable state to participate in a task transfer, either as a sender or a receiver. For example, a heavily loaded machine could try to start process migration when its load index exceeds a certain threshold.

4.2 Selection policy

This policy determines which task should be transferred. Once the transfer policy decides that a machine is in a heavily-loaded state, the selection policy selects a task for transferring. Selection policies can be categorized into two policies: preemptive and non-preemptive. A preemptive policy selects a partially executed task. Such, a preemptive policy should also transfer the task state which can be very large or complex. Thus, transferring operation is expensive. A non- preemptive policy selects only tasks that have not begun execution and, hence, it does not require transferring the state of task.

4.3 Location policy

The objective of this policy is to find a suitable transfer partner for a machine, once the transfer policy has decided that the machine is a heavily-loaded state or lightly-loaded one. Common location policies include: random selection, dynamic selection, and state polling.

4.4 Information policy

This policy determines when the information about the state of other machines should be collected, from where it has to be collected, and what information is to be collected.

5. LOAD BALANCING IN CLIENT SERVER SYSTEMS

When multiple web servers are present in a server group, the HTTP traffic needs to be evenly distributed among the servers. In the process these servers must appear as one web server to the web client, for example an internet browser. The load balancing mechanism used for spreading HTTP requests is known as IP spraying. The equipment use for IP spraying is also called the load dispatcher or network dispatcher or simply the load balancer. In this case the sprayer intercets each HTTP request, and redirects them to a server in the server cluster. Depending on the type of the sprayer involved, the architecture can provide scalability, load balancing and fail over requirements. Load balancing of servers by an IP sprayer can be implemented in different ways. These methods of load balancing can be set up in the load balancer based on available load balancing types. There are various algorithms used to distribute the load among the available servers.

5.1 Random Allocation

In a random allocation, the HTTP requests are assigned to any server picked randomly among the group of servers. In such a case, one of the servers may be assigned many more requests to process, while the other servers are sitting idle. However, on average, each server gets its share of the load due to the random selection.

Pros: Simple to implement.

Cons: Can lead to overloading of one server while under-utilization of others.

5.2 Round-Robin Allocation

In a round-robin algorithm, the IP sprayer assigns the requests to a list of the servers on a rotating basis. The first request is allocated to a server picked randomly from the group, so that if more than one IP sprayer is involved, not all the first requests go to the same server. For the subsequent requests, the IP sprayer follows the circular order to redirect the request. Once a server is assigned a request, the server is moved to the end of the list. This keeps the servers equally assigned.

Pros: Better than random allocation because the requests are equally divided among the available servers in an orderly fashion.

Cons: Round robin algorithm is not enough for load balancing based on processing overhead required and if the server specifications are not identical to each other in the server group.

5.3 Weighted Round-Robin Allocation

Weighted Round-Robin is an advanced version of the round-robin that eliminates the deficiencies of the plain round robin algorithm. In case of a weighted round-robin, one can assign a weight to each server in the group so that if one server is capable of handling twice as much load as the other, the powerful server gets a weight of 2. In such cases, the IP sprayer will assign two requests to the powerful server for each request assigned to the weaker one.

Pros: Takes care of the capacity of the servers in the group.

Cons: Does not consider the advanced load balancing requirements such as processing times for each individual request.

The configuration of a load balancing software or hardware should be decided on the particular requirement. For example, if the website wants to load balance servers for static HTML pages or light database driven dynamic webpages, round robin will be sufficient. However, if some of the requests take longer than the others to process, then advanced load balancing algorithms are used. The load balancer should be able to provide intelligent monitoring to distribute the load, directing them to the servers that are capable of handling them better than the others in the cluster of server.

6. LOAD BALANCING IN PEER TO PEER SYSTEMS

6.1 Static Algorithms

In static algorithms the tasks are assigned to various nodes before the start of their execution and cannt be preempted. The goal of static load balancing method is to reduce the overall execution time of a concurrent program while minimizing the communication delays. A general disadvantage of all static schemes is that the final selection of a host for process allocation is made when the process is created and cannot be changed during process execution to make changes in the system load.

Round Robin and Randomized Algorithms

In the round robin methods the processes are allocated evenly between all processors. Each new process is assigned to new processor in round robin order. The process allocation order is maintained on each processor locally independent of allocations from remote processors. With equal workload round robin algorithm is expected to work well. Round Robin and Randomized schemes [16] work well with number of processes larger than number of processors. Advantage of Round Robin algorithm is that it does not require inter-process communication. Round Robin and Randomized algorithm both can attain the best performance among all load balancing algorithms for particular special purpose applications. In general Round Robin and Randomized are not expected to achieve good performance in general case.

Central Manager Algorithm

In this algorithm, a central processor selects the host for new process [17]. The minimally loaded processor depending on the overall load is selected when process is created. Load manager selects hosts for new processes so that the processor load confirms to same level as much as possible. From the available information of the system load state, central load manager makes the load balancing judgment. This information is updated by remote processors, which send a message each time the load on them changes. The load on a system may change because of creation of a child process or because of termination of any process. The load manager makes load balancing decisions based on the system load information. The drawback here is the high degree of inter-process communication which could lead to a bottleneck state. This algorithm is expected to perform better than the parallel applications, especially when dynamic activities are created by different hosts.

Threshold Algorithm

According to this algorithm, the processes are assigned immediately upon creation to hosts. Hosts for new processes are selected locally without sending remote messages. Each processor keeps a private copy of the system's load. The load of a processor can be characterized by one of the three levels: Under loaded, medium and Over loaded. Two threshold parameters *tunder* and *tupper* can be used to describe these levels.

Under loaded - load < tunder
Medium - tunder ≤ load ≤ tupper
Overloaded - load > tupper

Initially, all the processors are considered to be under loaded. When the load state of a processor exceeds a load level limit, then it sends messages regarding the new load state to all remote processors, regularly updating them as to the actual load state of the entire system. If the local state is not overloaded then the process is allocated locally. Otherwise, a remote under loaded processor is selected, and if no such host exists, the process is also allocated locally. Thresholds algorithm have low inter process communication and a large number of local process allocations. The later decreases the overhead of remote process allocations and the overhead of remote memory accesses, which leads to improvement in performance. A disadvantage of the algorithm is that all processes are allocated locally when all remote processors are overloaded. A load on one overloaded processor can be much higher than on other overloaded

processors, causing significant disturbance in load balancing, and increasing the execution time of an application.

6.2 Dynamic Algorithms

It differs from static algorithms in that the work load is distributed among the processors at runtime. The processes are allocated to the various processors based on the latest information collected [8, 18]. Unlike static algorithms, dynamic algorithms allocate processes dynamically when one of the processors becomes under loaded. . Instead, they are buffered in the queue on the main host and allocated dynamically upon requests from remote hosts.

Central Queue Algorithm

Central Queue Algorithm [19] works on the principle of dynamic distribution. It stores new activities and unfulfilled requests as a cyclic FIFO queue on the main host. Each new activity arriving at the queue manager is inserted into the queue. Then, whenever a request for an activity is received by the queue manager, it removes the first activity from the queue and sends it to the requester. If there are no ready activities in the queue, the request is buffered, until a new activity is available. If a new activity arrives at the queue manager while there are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it. When a processor load falls under the threshold, the local load manager sends a request for a new activity to the central load manager. The central load manager answers the request immediately if a ready activity is found in the *process-request queue*, or queues the request until a new activity arrives.

Local Queue Algorithm

Main feature of the algorithm by ***Leinberger et al*** [19] is dynamic process migration support. The basic idea of the local queue algorithm is static allocation of all new processes with process migration initiated by a host when its load falls under threshold limit. The parameter, load, defines the minimal number of ready processes the load manager attempts to provide on each processor. Initially, new processes created on the *main* host are allocated on all under loaded hosts. The number of parallel activities created by the first parallel construct on the main host is usually sufficient for allocation on all remote hosts. From then on, all the processes created on the main host and all other hosts are allocated locally. When the host gets under loaded, the local load manager attempts to get several processes from remote hosts. It randomly sends requests with the number of local ready processes to remote load managers. When a load manager receives such a request, it compares the local number of ready processes with the received number. If the former is greater than the latter, then some of the running processes are transferred to the requester and an affirmative confirmation with the number of processes transferred is returned.

7. CONCLUSION

From this comparative study of various algorithms for Client Server and Peer to Peer systems we conclude that dynamic algorithms are better as compared to Static Algorithms in terms of better response time and throughput. But the drawback with them is that they need more computations and the process migration as well as state information collection could be a communication overhead. It has also been observed that fuzzy algorithms give better results as they take the global state uncertainty also into consideration.

8. REFERENCES

[1]. J. Li and H. Kameda, "Load Balancing Problems for Multiclass Jobs in Distributed/Parallel Computer Systems," *IEEE Trans. Computers*, vol. 47, no. 3, pp. 322-332, March 1998.

[2]. Z. Zeng and V. Bharadwaj, "A Static Load Balancing Algorithm via Virtual Routing," *Parallel and Distributed Computing and Systems*, Marina del Rey, CA, USA, pp. 244-249, November, 2003.

[3]. L. Anand, D. Those, V. Mani, ELISA: An estimated load information scheduling algorithm for distributed computing systems, Computers and Mathematics with applications 37 (1999) 57 - 85.

[4]. J.Watts , S. Taylor, A practical approach to dynamic load balancing, IEEE Transactions on Parallel and Distributed Systems 9 (3) (1998) 235-248.

[5]. G. Manimaran, C.Siva Ram Murthy, An efficient dynamic scheduling algorithm for multiprocessor real time systems, IEEE Transactions on Parallel and Distributed Systems 9 (3) (1998) 312-319.

[6]. Sameena Naaz, Afshar Alam, Ranjit Biswas "Implementation of a new Fuzzy Based Load Balancing Algorithm for Hypercubes" IJCSIS 2010

[7]. Ahmed and A. Ghafoor, "Semi-Distributed Load Balancing for Massively Parallel Multicomputers," IEEE Trans. Software Eng., vol. 17, no. 10, pp 987-1004, October 1991.

[8]. Y. Wang and R. Morris, "Load Sharing in Distributed Systems," IEEE Trans. Comput., vol. C-34, no. 3, pp. 204-217, Mar. 1985.

[9]. K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," IEEE Trans. Comput., vol. 38, no. 8, pp 1110-1123, August 1989

[10]. S. Sharma, S. Singh, and M. Sharma, "Performance Analysis of Load Balancing Algorithms," World Academy of Science, Engineering and Technology, vol. 38, 2008.

[11]. Zeng Zeng, Veeravalli, B., "Rate-based and queue-based dynamic load balancing algorithms in distributed systems", Proceedings of the Tenth International Conference on Parallel and Distributed Systems, ICPADS 2004.

[12]. I. Ahmad and A. Ghafoor, "A semi distributed task allocation strategy for large hypercube supercomputers," in Proc. Supercomputing' 90, pp. 898-907, 1990.

[13]. A. Kumar, M. Singhal, and M.T. Liu, "A model for distributed decision making: An expert system for load balancing in distributed systems," COMPSAC, pp. 507-513, 1987.

[14]. K.K. Goswami, M. Devarakonda, and R.K. Iyer, "Prediction- based dynamic load-sharing heuristics," IEEE Trans. Parallel Distrib. syst., vol. 4, pp. 638-648, June 1993.

[15]. Chulhye Park and Jon G. Kuhl, " A Fuzzy Based Distributed Load Balancing Algorithm for Large Distributed Systems", Proceedings of the Second International Symposium on Autonomous Decentralized Systems (ISADS'95).

[16]. R. Motwani and P. Raghavan, "Randomized algorithms", ACM Computing Surveys (CSUR), 28(1):33-37, 1996

[17]. P. L. McEntire, J. G. O'Reilly, and R. E. Larson, Distributed Computing: Concepts and Implementations. New York: IEEE Press, 1984.

[18]. S. Malik, "Dynamic Load Balancing in a Network of Workstation", 95.515 Research Report, 19 November, 2000.

[19]. William Leinberger, George Karypis, Vipin Kumar, "Load Balancing Across Near-Homogeneous Multi-Resource Servers", Proceedings of the 9th Heterogeneous Computing Workshop, ISBN: 0-7695-0556-2.