# A Guide to Dynamic Load Balancing in Distributed Computer Systems

**Ali M. Alakeel**

College of Computing and Information Technology
University of Tabuk, Tabuk, Saudi Arabia

**Summary**
Load balancing is the process of redistributing the work load among nodes of the distributed system to improve both resource utilization and job response time while also avoiding a situation where some nodes are heavily loaded while others are idle or doing little work. A dynamic load balancing algorithm assumes no a priori knowledge about job behavior or the global state of the system, i.e., load balancing decisions are solely based on the current status of the system. The development of an effective dynamic load balancing algorithm involves many important issues: load estimation, load levels comparison, performance indices, system stability, amount of information exchanged among nodes, job resource requirements estimation, job's selection for transfer, remote nodes selection, and more. This paper presents and analyses the aforementioned issues that need to be considered in the development or study of a dynamic load balancing algorithm.

***Keywords:***
*Distributed computer systems; communication networks; load balancing; load sharing; performance evaluation; stability.*

## 1. Introduction

In a distributed computer system environment, as described in [1], where two or more autonomous computers are connected via a communication network, resource sharing is a most desirable feature. Apart from sharing data and I/O devices, nodes of a distributed system could further improve system performance by sharing their computational power. Load balancing is a mechanism that enables jobs to move from one computer to another within the distributed system. This creates faster job service e.g., minimize job response time[1] and enhances resource utilization. Various studies, e.g., [2]-[18], have shown that load balancing among nodes of a distributed system highly improves system performance and increases resource utilization.

Load balancing is the process of roughly equalizing the work load among all nodes of the distributed system. It strives to produce a global improvement in system performance. In this manner, load balancing goes one step further than load sharing, e.g., [6], [19], [20], which only avoids having some nodes idle in the distributed system when other nodes have too much work [13]. Load balancing has been found by [21] to further reduce the mean and standard deviation of task response times more than load sharing would.

Some of the main goals of a load balancing algorithm, as pointed out by [8] are: (1) to achieve a greater overall improvement in system performance at a reasonable cost, e.g., reduce task response time while keeping acceptable delays; (2) to treat all jobs in the system equally regardless of their origin; (3) to have a fault tolerance: performance endurance under partial failure in the system; (4) to have the ability to modify itself in accordance with any changes or expand in the distributed system configuration; and (5) maintain system stability: the ability to account for emergency situations such as sudden surge of arrivals so that system performance does not deteriorate beyond a certain threshold while preventing nodes of the distributed system from spending too much time passing up jobs among themselves instead of executing these jobs.

The development of an effective dynamic load balancing algorithm involves the consideration of many important issues. This paper presents and analyses the most important issues which need to be considered in the development of an effective load balancing algorithm: load estimation, load levels comparison, performance indices, stability, amount of information exchanged among nodes, job resource requirements estimation, jobs selection for transfer, remote nodes selection, and more. Our objective is to provide a guide of the critical issues that need to be addressed in the development or study of a dynamic load balancing algorithm.

[1] The time a job spends waiting for service plus service time.

Manuscript received June 5, 2010
Manuscript revised June 20, 2010

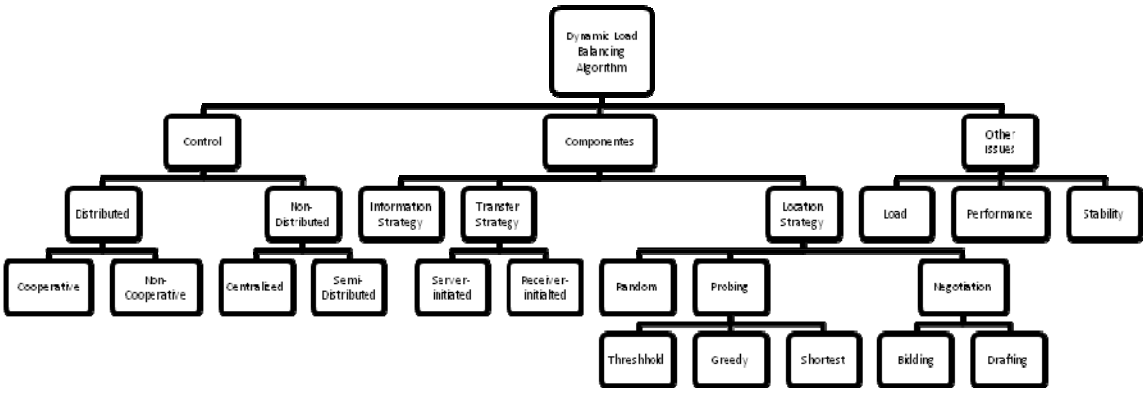


Fig. 1 Important issues of dynamic load balancing algorithms.

## 2. Load Balancing

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. Consequently, a vast amount and variety of research has been conducted in an attempt to solve this problem. This section presents some of the most important techniques and approaches previously employed to achieve load balancing in a distributed system. We will only refer to a specific algorithm when necessary to further clarify our explanation. Taxonomies of load balancing algorithms in distributed systems are reported in [8], [20], and [22].

Solutions to the load balancing problem are divided into two main approaches depending on whether a load balancing algorithm bases its decisions on the current state of the system or not: static and dynamic.

In the static approach, e.g., [3], [12], [24], [25], [27], priori knowledge about the global status of the distributed system, job resource requirement, and communication time are assumed. In this approach, load balancing is achieved by providing a mapping or assignment from a set of tasks to a set of processors such that a performance's function is minimized. Although this assignment can take either a deterministic or a probabilistic form, the current state of the system is not considered in either of them [6]. In a deterministic assignment, for instance, node i ships extra tasks to node j all of the time. In a probabilistic assignment, however, node i sends extra tasks to node l with probability p and to node m with probability q. The major drawback of the static approach is that it does not take the current state of the system into account when making these decisions. This has a major impact on the overall system performance due to the unpredictability of load fluctuation of the distributed system. Some techniques employed in static load balancing are: solution space enumeration and search, graph theoretic, mathematical programming, and queuing theoretic [22].

In the dynamic approach, e.g., [4], [6], [10], [14], [15], [18], [26], [28]-[38], load balancing decisions are 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 [6], [8]. For this reason, we will focus our attention on dynamic load balancing algorithms in this research. Hence, static load balancing will not be discussed any further.

### 2.1 Dynamic Load Balancing

This section presents some of the important issues related to dynamic load balancing in distributed systems. We also discuss some of the different approaches previously used to handle each of these issues as reported in the literature. Fig. 1 portrays the structure we follow in our discussion of important issues concerning a dynamic load balancing algorithm.

#### 2.1.1 The Responsibility of Control

The control mechanism used to derive a dynamic load balancing algorithm affects system performance in two

main areas: (1) overhead introduced by the algorithm and (2) system fault tolerance. Obviously, a load balancing algorithm that requires too many messages in order to reach its decisions is not desirable. Similarly, a dynamic load balancing algorithm which does not have precautions for the halt of one or more of its components is not desirable.

In a distributed system, dynamic load balancing can be carried out in two different schemes: distributed and non-distributed. In a distributed scheme, e.g., [4], [14], [18], [28], [29], [31, [37], the dynamic load balancing algorithm is executed by all nodes in the system and the responsibility of load balancing is shared among them. The interaction among nodes to achieve load balancing can take two forms: cooperative and non-cooperative. In a cooperative form, nodes work together to achieve a global objective, e.g., to improve the system's overall response time. In a non-cooperative form, each node works independently toward a local goal, e.g., to improve a local task's response time.

Distributed dynamic load balancing algorithms tend to generate more messages than non-distributed algorithms. This is due to the fact that each node might need to interact with all other nodes in the system in order to make its load balancing decisions. An advantage, however, is that the failure of one or more nodes in the system will not cause the whole operation of load balancing to halt; it only partially degrades system performance.

Although the majority of dynamic load balancing algorithms proposed in the literature are distributed, it does not mean that the distributed control is effective in all of them. For those algorithms that require each node to exchange status information with every other node in the network, distributed control could be a great burden on the communication system which affects the overall system performance negatively. Distributed control is of the greatest advantage when each node is given the maximum chance to act alone or to interact with as few nodes as possible. Needless to say, most proposed dynamic load balancing algorithms require full interaction among nodes of the distributed system. Hence, there is a great need for distributed dynamic load balancing algorithms that call for minimum interaction among nodes.

In a non-distributed scheme, the responsibility of load balancing is either taken on by a single or some nodes but never with all nodes. Non-distributed based dynamic load balancing can take two forms: centralized and semi-distributed. In a centralized form, e.g., [12], [40], the load balancing algorithm is only executed by one node of the distributed system: the central node. The central node is solely responsible for load balancing of the whole distributed system. Other nodes in the distributed system react with the central node but not with each other. In a semi-distributed form, e.g., [39], nodes of the distributed system are segmented into clusters. Load balancing within each cluster is centralized; a central node is nominated to take charge of load balancing within this cluster. Load balancing of the whole distributed system is achieved through the cooperation of the central nodes of each cluster, i.e. the responsibility is distributed among the central nodes of each cluster. This approach was suggested in [39] to fit distributed systems with a large number of nodes.

Centralized dynamic load balancing requires fewer messages to reach a load balancing decision. This is because other nodes in the system do not interact with each other; they only interact with the central node. On the other hand, centralized algorithms jeopardize system performance in the event that the central node crashes. Also, there is a possibility that this node could cause a bottleneck if it became swamped with messages from all the other nodes in the system. A study by [17] has shown that centralized load balancing suits small sized networks (less than 100 nodes) more than any other control method.

### 2.1.2 Components of a Dynamic Load Balancing Algorithm

A dynamic load balancing algorithm is required to make load distribution decisions based on the current work load at each node of the distributed system. Consequently, this algorithm must provide a mechanism for collecting and managing system status information. The part of a dynamic load balancing responsible for collecting information about nodes in the system is referred to as information strategy in the literature. Also, a dynamic load balancing algorithm must include a mechanism to assist each node in deciding which job is eligible for load balancing. The part of a dynamic load balancing algorithm which selects a job for transfer from a local node to a remote node is referred to as transfer strategy. Furthermore, a dynamic load balancing algorithm must provide a mechanism on which a destination node for a transferred job is determined. The part of a dynamic load balancing algorithm which selects a destination node for a transferred task is referred to as location strategy.

Therefore, a dynamic load balancing algorithm has three main components: the information, transfer, and location strategies. Each of these strategies will be discussed in more detail later. As shown in Fig. 2, incoming jobs are intercepted by the transfer strategy which decides whether or not it should be transferred to a remote node for the purpose of load balancing. If the transfer strategy decides that a job should be transferred, the location strategy is triggered in order to find a remote node for the job. Information strategy provides both transfer and location strategies with the necessary information to build their decisions.

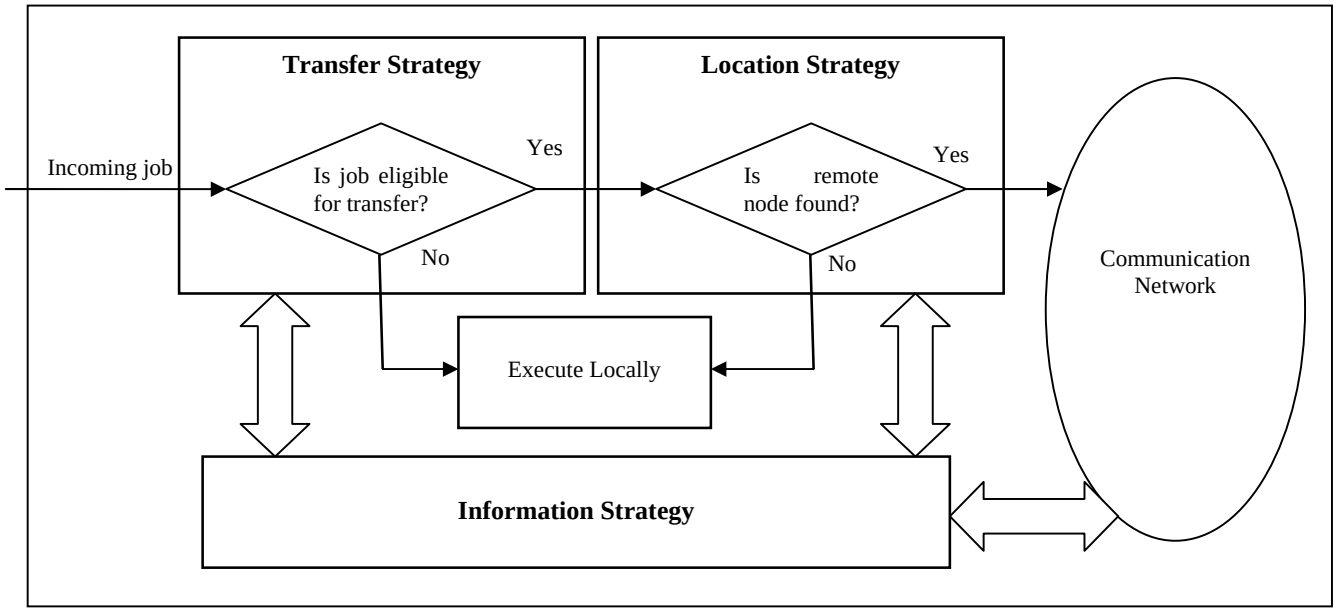


Fig. 2 Interaction among components of a dynamic load balancing algorithm.

2.1.2.1 Information Strategy:

Information strategy is the information center of a dynamic load balancing algorithm. It is responsible for providing location and transfer strategies at each node with the necessary information needed to make their load balancing decisions. A sophisticated information strategy keeps all nodes of the distributed system updated on the global system state but generates extra traffic and hence increases the overhead generated by the algorithm. Therefore, there is a trade-off between the amount of information exchanged and the frequency of the exchange of this information.

Some studies of dynamic load balancing algorithms based on the amount of information they use in order to make a load balancing decision are reported in [6] and [20]. It was concluded by [6] that algorithms which attempt to collect detailed information about system state in order to make the best decisions do not produce a significant performance gain over that produced by an algorithm which uses very little or no information at all. For example, the performance of a random algorithm which only used local information was close in performance to another algorithm which tried to utilize some global information.

2.1.2.2 Transfer Strategy:

Considering that important parameters such as job execution time, size, I/O, and memory requirements are not known until the job is executed, selecting a job for load balancing is not an easy task. More than one approach has been tried in order to deal with this missing information.

One approach to load balancing makes job transfer decisions independently of a job's characteristics. In this scheme, a job is transferred if the queue length at the local node exceeds a certain threshold. Otherwise, the job is executed locally. The main advantage of this approach is its generality, i.e. it is not directed toward a certain system. However, the inflexibility to discriminate among different sized jobs is a drawback. Examples of load balancing algorithms based on this approach may be found in [5], [6], and [43]. Different approach, e.g., [17], [19], [41], uses trace information about a job's behavior which is collected from a real system under study and employs this information to estimate job behavior in the future. Although this approach enhances the selections of an appropriate job for load balancing, the outcome result is only valid for the system under study and under a comparable load conditions. For instance, [19], selects jobs for load balancing based on their future resource requirements which is estimated using a statistical system developed in [42]. Also, the approach reported in [41] utilizes a history of job's execution times to differentiate between big and small sized jobs. The objective of this filtering mechanism is to enforce small sized jobs to be executed locally.

A third approach to load balancing, as described in [10], employs an automated tool to estimate future job's execution times. This is achieved through an on-line trace of each job's behavior under different load conditions. This information is then utilized to estimate a job's

execution time in the near future. The main advantage of this approach is its independence of the system under study, but the extra overhead associated with this approach is a problem.

Two main issues concerning load balancing activity that depend on the transfer strategy employed are: (1) when is the right time to start it and (2) what jobs are subjected to it. Two approaches are commonly used to start the load balancing activity: the time a new job arrives or is created at a node and the time a finished job departs from a node. Algorithms which make load balancing decisions at the arrival or creation of a new job are referred to as sender-initiated, while algorithms which make load balancing decisions at the departure of a finished job are referred to as receiver-initiated. It has been conceived that under sender-initiated strategy an overloaded node launches the load balancing activity in an attempt to get some other node in the network to accept some of its load. While under a receiver-initiated strategy an under-loaded node offers its willingness to accept more load. It has been shown in [43] that sender-initiated algorithms are suitable when the system is light to moderately loaded while receiver-initiated algorithms are suitable when the system is heavily loaded. This is assuming that job transfer cost under both strategies is comparable.

Two approaches that determine which jobs are eligible for transfer are: consider-new-only and consider-all. The consider-new-only approach, only considers newly arrived or created jobs for load balancing. This approach is commonly used, e.g., [5], [6], [28], [40], because of its simplicity. The consider-all approach, e.g., [19], considers all jobs eligible for load balancing. This approach is more complex than the previous one because it employs an extra mechanism for the selection of the appropriate job out of a set of active jobs. According to [19], consider-all approach performs better than consider-new-only when the size of jobs and resource requirements differ greatly.

### 2.1.2.3 Location Strategy:

One of the main decisions performed by a load balancing algorithm is the selection of a destination node for a job transferred for load balancing. This decision represents the sole purpose for load balancing: a heavily loaded node tries to find a lightly loaded node to help in executing some of its jobs. This decision is performed by the location strategy. The selection of a remote node is based on the current work load present at that node. Until we discuss different load measurements later, the load of a node is expressed as the CPU queue length (the number of jobs waiting for service plus the one in service). The amount of information used by the location strategy to select a destination node is a very important issue as we will see in the next paragraphs when we discuss some location strategies. Some of the approaches used to select a destination node for a transferred job are: random, probing and negotiation.

**Random.** Under a random location strategy, a local node selects a remote node randomly and transfers the job there for execution [6], [43]. Upon receiving this job, the remote node executes it if its load, i.e., queue length, is below a predefined threshold. Otherwise, this remote node will select a new destination node for this job. To avoid having this job ping ponged among nodes without getting serviced, a limit on the number of hops it could take is imposed which enforces the last node to receive that job to execute the job when this limit is reached regardless of its current load.

As shown in [6], the performance of this simple location strategy, which does not employ any information in its selection, was significant as compared to a system with no load balancing at all. The performance of this strategy is usually used as a reference point to compare other load balancing algorithms that collect global information.

**Probing.** Location strategies which employ probing work as follows: a local node selects a random subset of nodes and polls them to find a suitable destination node for a transferred job. A suitable node is the one which will provide the transferred task with a better service, i.e., better response time, than the local load from where it originated. To further clarify this concept, we present three location strategies: threshold [6], greedy [5], and shortest [6].

Under the threshold strategy, a local node selects a remote node at random and probes it to see if transferring a job to that node will cause its load to go above a threshold. If not, the job is transferred to this remote node; otherwise another remote node is selected at random and probed as before. The algorithm imposes a limit on the number of times a local node is allowed to do the probing. After that limit, the job is executed locally. As pointed out by [6], although the threshold strategy uses a small amount of information, it provides a substantial performance improvement as compared to the random location strategy.

A variation of the threshold strategy, the greedy strategy, was reported in [5]. The greedy strategy uses a cyclic probing of nodes instead of random probing used by the threshold. According to [CHOW90], the greedy strategy outperforms the threshold strategy. The good performance of the greedy strategy was not attributed to the probing mechanism alone, but also to the transfer strategy used.

The shortest location strategy selects a subset of remote nodes randomly and probes them to find out their current load, i.e. queue length. The remote node with the smallest queue length is then selected. If this selected node's queue length is smaller than a certain threshold, then the job is transferred there, otherwise the job is executed locally. Although the shortest strategy attempts to make a wiser selection than the threshold does, it is shown in [6]

that there was not a significant gain in performance over what has been achieved by the threshold strategy.

**Negotiation.** Under this location strategy, which is usually applied in distributed dynamic algorithms, nodes negotiate with each other for load balancing purposes in order to select a suitable destination node for transferred jobs. To further clarify how negotiations work, we present two location strategies: bidding, e.g., [14] and drafting, e.g., [30], which are based on this concept.

In a bidding location strategy, a heavily loaded node is the one who initiates load balancing. Hence, this implies that this strategy is coupled with a sender-initiated transfer strategy. Depending on the load estimation mechanism used, when a node gets overloaded it broadcasts a request-for-bid message to all other nodes in the network. A request-for-bid message includes information about the current load of the original node and information about the jobs this node is willing to ship abroad. Upon receiving a request-for-bid message, a remote node inspects the content of the message and compares its content with its own current status. If this remote node's work load is lighter than the load of the one who originated the request-for-bid message, this remote node will reply with a bid-message. Otherwise, the remote node just ignores the request-for-bid message (in some variation algorithms, the remote node returns its current load information without submitting a bid). A bid message includes the remote node's current load and other information specifying the amount of extra load this node could accommodate. After receiving all bid messages, if it is still overloaded, the original node selects the remote node with the best bid, i.e., the one having the lowest load, and transfers some of its load there. The major problem with the bidding strategy is that a lightly loaded node might get overwhelmed with work as a result of it winning many bids. Imposing a limit in the number of bids accepted could take care of this problem.

In a drafting location strategy, a lightly loaded node is the one who initiates load balancing. Hence, this implies that this strategy is coupled with a receiver-initiated transfer strategy. Under the drafting location strategy, nodes of the distributed system are grouped dynamically into three different groups according to their current load. A node could be in one of three states: lightly loaded (L-load ), neutrally loaded (N-load), or heavily loaded (H-load). Each node monitors its own load and periodically changes its state accordingly. After each change, each node broadcasts its state to all other nodes in the network. Each node keeps a table of all nodes status.

Under the drafting strategy when a node finds itself in L-load state it identifies all nodes in the H-state and sends a draft-request message to each of them in which the L-load node indicates its eagerness to accept more work load. Upon receiving the draft-request, a remote (drafted) node replies by sending a draft-response message only if it still in the H-load state. A draft-response message contains information about jobs eligible for transfer at the drafted node. When the original node receives all draft-response messages or after a time-out is reached, it selects a remote node based on a certain criterion and informs the selected (drafted) remote node of this decision by sending it a draft-select message. If still in H-state by the time it receives a draft-select message, the drafted node transfers some of its work to the original node. According to [30], the drafting strategy outperforms the bidding strategy when compared in the same environment.

### 2.1.3 Other Issues

Dynamic load balancing development involves many parameters and concerns. Load measurement and system performance evaluation are some of used by a dynamic load balancing algorithm. An important outcome of a dynamic load balancing algorithm that is of a great concern to the developer is whether the algorithm is stable or not. This section highlights some important issues related to parameters and stability measures in dynamic load balancing.

**Load Measurement.** As discussed in previous sections, most decisions made by a dynamic load balancing algorithm depend on the current work load in the system. For this reason, one of the most important parameters used by a dynamic load balancing algorithm is the load descriptor it employs to define the work load present at each node of the system. Some load descriptors are: CPU queue length, CPU utilization, job resource requirements, context switch rate, percentage of idle CPU time, and the amount of unfinished work at a node.

CPU queue length is believed to be a good load descriptor because it gives a good estimate of job response time. It has been the most commonly used load descriptor employed by dynamic load balancing algorithms. The advantage of queue length as a load descriptor is the simplicity to obtain its value. Despite that, [19] showed that job resource requirement is a better load descriptor when there is a mechanism to predict this value in advance and jobs are then served in round-robin fashion. Round-robin scheduling treats the CPU queue as a circular queue. The CPU scheduler goes around the ready queue allocating the CPU to each job for a time interval (time quantum).

**Performance's Measurements.** The ultimate objective of a dynamic load balancing algorithm is to improve system performance. Therefore, a load balancing algorithm should adopt a performance index by which this performance improvement is measured. Since there is more than one index that can be utilized, the selection usually differs from one algorithm to another.

A performance index could be system performance-oriented, user-oriented, or both [8]. System throughput and resource utilization are examples of system-oriented performance indices. Mean response time of the distributed system and job mean execution time are user-oriented performance indices. Other performance indices such as job mean wait time, mean and standard deviation of a job wait time, and a job wait ratio (the wait time per unit of service) could be used to reflect system's performance user expectation. System mean response time is the performance index that is commonly used by the majority of load balancing algorithms.

**System's Stability.** Like any dynamic system, system stability is of a major concern in dynamic load balancing algorithms. It is very important that a dynamic load balancing algorithm maintain stability in the distributed system.

A load balancing algorithm is stable if it: (1) does not cause nodes of the system to enter a state of processor thrashing (the state where nodes spend all their time in passing jobs among themselves without getting these jobs executed) [4], [6]; (2) if the load on any two nodes of the distributed system does not differ by more than a certain percentage x; and (3) if the response time to any sudden arrival burst does not exceed a certain limit [8], [22].

## 3. Conclusion

In this paper an extensive review of the most important issues related to the development of dynamic load balancing algorithms for multicomputer distributed systems was presented. Load estimation, load levels comparison, performance indices, stability, amount of information exchanged among nodes, job resource requirements estimation, job selection for transfer, remote nodes selection, are some of the issues that have been discussed. Our objective is to provide a guide to the critical issues that need to be addressed while the development or study of a dynamic load balancing algorithm.

## References


[1] P. Enslow Jr., "What is a "Distributed" Data Processing System?" Computer, Vol. 11, No. 1, pp. 13-21, January 1978.

[2] Z. Khan, R. Singh, J. Alam, and R. Kumar, "Performance Analysis of Dynamic Load Balancing Techniques for Parallel and Distributed Systems," International Journal of Computer and Network Security, vol. 2, no. 2, February 2010.

[3] X. Tang and S.T. Chanson, "Optimizing Static Job Scheduling in a Network of Heterogeneous Computers," Proc. of the Intl. Conf. on Parallel Processing, pp. 373-382, August 2000.

[4] R. M. Bryant and R. A. Finkel, "A Stable Distributed Scheduling Algorithm," in Proc. 2nd Int. Conf. Dist. Comp., pp. 341-323, April 1981.

[5] S. Chowdhury, "The Greedy Load Sharing Algorithms," J. Parallel and Distributed Comput., vol. 9, pp. 93-99, May 1990.

[6] D.L. Eager, E.D. Lazowski, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," IEEE Trans. Software Eng., vol. SE-12, no. 5, pp. 662-675, May 1986.

[7] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," Computer, vol. 15, no. 6, pp. 50-56, June 1982.

[8] A. Goscinski, "Distributed Operating Systems," Addison-Wesley, Sydney, 1991.

[9] M. Livny and M. Melman, " Load Balancing in Homogeneous Broadcast Distributed Systems. In Proc. ACM Comput. Network Performance Symp., pp. 47-55, 1982.

[10] 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.

[11] R. Mirchandaney, D. Towsley, and J. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems," Journal of Parallel and Distributed Computing, No. 9, pp. 331-346, 1990.

[12] L. Ni, and K. Hwang, K., "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," IEEE Transactions on Software Engineering, Vol. SE-11, pp. 491-496, May 1985.

[13] N. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems. Computer, Vol. 25, No. 12, p.33-44, December 1992.

[14] J. A. Stankovic and I. S. Sidhu, "An Adaptive Bidding Algorithm for Processes, Cluster and Distributed Groups," in Proc. 4th Int. Conf. Distributed Compu. Sys., pp. 49-59, 1984.

[15] J. Stankovic, "Simulations of Three Adaptive, Decentralized Controlled, Task Scheduling Algorithms," Computer Networks, Vol. 8, No. 3, pp. 199-217, June 1984.

[16] H. S. Stone, "High-Performance Computer Architecture," 2nd ed., Addison Wesley, Reading, MA, 1990.

[17] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing. IEEE Transactions on Software Engineering, Vol. SE-14, No. 9, pp. 1327-1341, September 1988.

[18] A. Barak and A. Shiloh, "A Distributed Load-balancing Policy for a Multicomputer," Software-Practice and Experience, Vol. 15, No. 9, pp. 901-913, September 1985.

[19] K. Goswami, M. Devarakonda, and R. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristics," IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 6, pp. 638-648, June 1993.

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

[21] P. Kruger, P. and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies," Proceedings of the Seventh International Conference in Distributed Computing Systems, pp. 242-249, 1987.

[22] T. L. Casavant, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," IEEE Trans. Software Eng., vol 14, no. 2, pp 141-154, February 1988.

[23] C. Kim and H. Kameda, "An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems," IEEE Trans. Comput., vol. 41, no. 3, pp. 381-384, March 1992.

[24] H. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, pp. 85-93, January 1977.

[25] A. N. Tantawi and D. Tawsley, "Optimal Static Load Balancing in Distributed Computer Systems," J. of Assoc. Comput., vol. 32, no. 2, pp. 445-465, April 1985.

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

[27] S. H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," IEEE Trans. Software Eng., vol. SE-5, no. 4, pp. 341-439, July 1979.

[28] D. Evans, D. and W. Butt, “Dynamic Load Balancing Using Task-Transfer Probabilities,” Parallel Computing, Vol. 19, pp 897-916, 1993.

[29] R. Mirchandaney and J. Stankovic, “Using Stochastic Learning Automata for Job Scheduling in Distributed Processing Systems,” Journal of Parallel and Distributed Computing, Vol. 3, pp. 527-552, 1986.

[30] L. Ni, C. Xu, and T. Gendreau, “A Distributed Drafting Algorithm for Load Balancing,” IEEE Transactions on Software Engineering, Vol. SE-11, No. 10, pp. 1153-1161, October 1985.

[31] J. Stankovic, “Bayesian Decision Theory and Its Application to Decentralized Control of Task Scheduling,” IEEE Transactions on Computers, Vol. C-34, No. 2, pp. 117-130, ,February 1985.

[32] S. Penmasta and A. T. Chronopoulos, "Dynamic Multi-User Load Balancing in Distributed Systems", 2007 IEEE International Parallel and Distributed Processing Symposium, pp. 1-10, Long Beach, CA, USA, March 2007.

[33] L. M. Campos and I. Scherson, "Rate of Change Load Balancing in Distributed and Parallel Systems," Parallel Computing, vol. 26 no. 9, pp. 1213-1230, July 2000.

[34] C.C. Hui and S. T. Chanson, "Improved Strategies for Dynamic Load Balancing," IEEE Concurrency, vol. 7, no. 3, pp. 58-67, July-Sept., 1999.

[35] A. Corradi, L. Lenoardi, and F. Zamboelli, "Diffusive Load Balancing Policies for Dynamic Applications," IEEE Concurrency, vol. 7, no. 1, pp. 22-31, Jan-March, 1999.

[36] S. Dhakal, M. M. Hayat, J.E.Pezoa, C. Yang, and D. Bader, "Dynamic Load Balancing in Distributed System in the Presence of Delays: A Regeneration-Theory Approach,", IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 4, April 2007.

[37] D. Grosu and A. T. Chronopoulos," Noncooperative Load Balancing in Distributed Systems," Journal of Parallel and Distributed Computing, vol. 65, no. 9, pp. 1022-1034, Sept. 2005.

[38] Z. Zeng and B. Veeravalli, "Rate-based and Queue-based Dynamic Load Balancing Algorithms in Distributed Systems," Proc. of 10th Int. Conf on Parallel and Distributed Systems, pp. 349-356, July 2004.

[39] I. 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.

[40] Y. Chow and W. Kohler, “Models for Dynamic Load Balancing in Heterogeneous Multiple Processor System,” IEEE Transactions on Computers, Vol. C-28, pp. 354-361, , May 1979.

[41] A. Svensson, History, “An Intelligent Load Sharing Filter,” Proceedings of the 10th International Conference in Distributed Computing Systems, pp. 546-553, May 1990.

[42] M. Devarakonda and R. Iyer, “Predictability of Process Resource Usage: A measurement-Based Study on Unix,” IEEE Transactions on Software Engineering, Vol. 15, No. 12, pp. 1579-1586, December 1989.

[43] D. Eager, E. Lazowski, and J. Zahorjan, “A Comparison of Receiver-Initiated and Sender Initiated Adaptive Load Sharing,” Performance Evaluation, Vol. 6, pp. 53-68, March 1986.

**Ali M. Alakeel** (also known as Ali M. Al-Yami) obtained his PhD degree in computer science from Illinois Institute of Technology, Chicago, USA in Dec. 1996, his M.S. degree in computer science from University of Western Michigan, Kalamazoo, USA in Dec. 1992 and his B.Sc. degree in computer science from King Saud University, Riyadh, Saudi Arabia in Dec. 1987. He is now with the College of Computing and Information Technology, University of Tabuk, Saudi Arabia. His current research interests include automated software testing, distributed computing, cellular networks, and fuzzy logic.