VOL. 12, NO. 3, FEBRUARY 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

SIGNIFICANT FACTORS IN THE DESIGN OF AN EFFICIENT DYNAMIC LOAD BALANCING ALGORITHM: AN EXPLORATION

V. Anand[1], Narasimhan Renga Raajan[2] and K. Anuradha[2]
[1]School of Computing, SASTRA University, Tirumalaisamudram, Thanjavur, Tamil Nadu, India
[2]School of Electrical and Electronics Engineering, SASTRA University, Tirumalaisamudram, Thanjavur, Tamil Nadu, India
E-Mail: anandwithah@gmail.com

ABSTRACT

This paper aims at studying various algorithms for dynamic load balancing pertinent to the significant issues handled by them. When some of the nodes are overloaded and other nodes are moderately or even under loaded, the process of load balancing redistributes the work load. By this means, utilization of resources and response time can be enhanced. Numerous algorithms are available for dynamic load balancing. Based on the current position of the system, these algorithms make the load balancing decisions. Several factors such as performance indices, load estimation, amount of information exchanged among nodes, load levels comparison, system stability, choosing remote nodes, and estimation of resource requirements should be taken into account. Consideration of these factors contributes a lot for developing an efficient dynamic load balancing algorithm. Relevant to the design of efficient algorithms for dynamic load balancing, this work brings into limelight the aforesaid factors.

Keywords: load balancing, distributed computer systems, load distribution, performance assessment, stability.

INTRODUCTION

Sharing of resources is a necessary requirement in a distributed computer system environment. As defined by P. Enslow Jr. [1], in a distributed computer system environment, many independent systems are linked through a communication network. To enhance the performance, systems should share the power of computation in addition to sharing the other resources such as devices and data.

To improve the resource utilization and reduce the response time, load balancing empowers the jobs to be shifted from one system to another. Lot of work has been done relevant to the load balancing in a distributed setup. The work proposed in [2]-[18] clearly illustrates that load balancing between systems in a distributed setup greatly improves the utilization of resources and also enhances the performance. By means of load balancing, the work load is equally distributed between systems in a distributed environment. This will end up in improving the performance of the system globally.

Improving the system performance globally is a merit of load balancing which cannot be done easily with load sharing. In the distributed system, the work proposed in [6], [19], [20] deals with systems, where certain nodes are idle whereas other nodes are overloaded. But, the work offered by P. Kruger and M. Livny [21] deals with load balancing. P. Kruger and M. Livny [21] have presented methods which minimize the standard deviation and mean of response time than load sharing schemes.

As specified by A. Goscinski [8], the objectives of a load balancing algorithm are (1) to accomplish a globally improved performance of the system (2) to give equal consideration for all jobs in the system (3) there should be a fault tolerance strategy if there is a partial failure (4) capable of adapting to changes in the distributed setup (5) preserve the steadiness of the system. Substantially

For developing an efficient dynamic load balancing algorithm, several factors are to be considered. This paper presents an insight into the following factors for designing an efficient dynamic load balancing algorithm: performance indices, load estimation, amount of information exchanged among nodes, load levels comparison, system stability, choosing remote nodes, and estimation of resource requirements. The main aim of this paper is to present an idea about these factors that should be focused while developing an efficient algorithm for dynamic load balancing.

An outline of load balancing

Load balancing has garnered significance and focus because of the rapid developments in distributed systems. Relevant to load balancing, lot of research were done.

This section offers the key methods proposed to accomplish load balancing in a distributed setup. To explain the methods with much clarity, this study will mention specific algorithms only on demand. A clear categorization of algorithms for load balancing in distributed systems were proposed in [8], [20], and [22].

Algorithms for load balancing are of two types, viz., static and dynamic. They are distinguished by the way that, decisions are taken based on the current state of the system (dynamic) or not (static). The work proposed in [3], [12], [24], [25], [27] have presented static methods. These static methods should have prior information about the job resource requirement, status of the distributed system as a whole and communication time. To accomplish load balancing, these methods allocate a set of tasks to a set of processors. In the work proposed by D.L. Eager et al. [6], these allocations were done by means of probabilistic or deterministic methods without considering the present state of the system. In the case of assignments using deterministic methods, always node i assigns additional jobs to node j. But, when using a probabilistic

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

method, node i assigns additional tasks to node j with probability p and to node k with probability q. The main problem with static methods is, the current state is not considered while taking decisions about the assignment.

When using static methods, the performance of the system is greatly affected because of the irregular changes in the load. Certain static methods applied for load balancing quoted by T. L. Casavant [22] are queuing theoretic, graph theoretic, solution space enumeration and search, and mathematical programming.

Some of the dynamic methods are presented in [4], [6], [10], [14], [15], [18], [26], [28]-[38]. In dynamic methods, decisions for load balancing are taken dynamically (i.e.) tasks are shifted from an overloaded node to an under-loaded node. The key advantage of dynamic method is it is capable of reacting to changes in the system.

When comparing the static and dynamic methods for load balancing, attaining a solution in a dynamic method is difficult than attaining a solution in a static method. Since the decisions for load balancing are taken relying on the current work load, dynamic methods can yield a better performance as reported by D.L. Eager et al. [6] and A. Goscinski [8]. Designing an efficient algorithm always needs methods with improved performance. So, this survey explores the algorithms for dynamic load balancing.

Factors and methods for dynamic load balancing

Relevant to dynamic load balancing, this section offers significant factors. This section also highlights various methods which has handled these factors. A clear structure of major factors deliberately discussed in this paper is depicted by Figure-1.

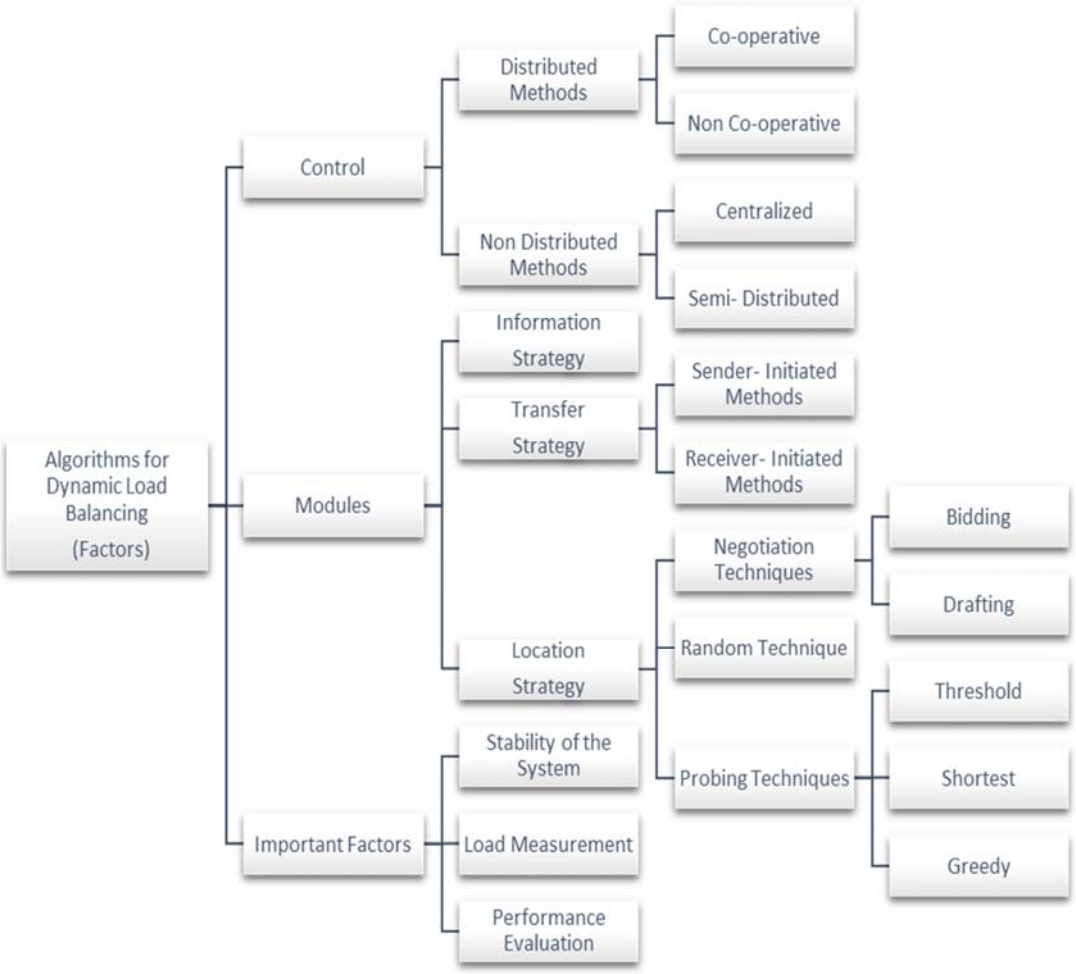

Figure-1. Significant factors for an efficient dynamic load balancing algorithm.

The duty of control

While obtaining the results with a dynamic load balancing algorithm, the control mechanism may deteriorate the performance of the system by two means:

a) an additional overhead added by the algorithm and

b) fault tolerance of the system.

Two undesirable things for a load balancing algorithm are it should not need lot of messages to take

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

decisions and the algorithm which does not have alternate provisions for the repair of some of its components.

Dynamic load balancing can be done by means of two dissimilar methods: distributed and non-distributed. The distributed schemes proposed in [4], [14], [18], [28], [29], [31, [37] were implemented in such a way that all the nodes executes the algorithm and the nodes share the duty of balancing the work load. In distributed schemes, the nodes interact with each other through co-operative or non-cooperative method. In a co-operative method, the overall response time of the system is enhanced (i.e.) global. In a non-cooperative method, response time of the local task is improved (i.e.) local objective of each node.

A problem with distributed algorithms is it generates lot of messages than non-distributed. The reason is each node has to communicate with all other nodes to make decisions for load balancing. But, still this is a gain. Because, when some of the nodes fail, it will not affect the process of load balancing. It will partially reduce the performance only.

Distributed control is not efficient in all the dynamic algorithms for load balancing. Distributed control could be an overhead when each of nodes in the system has to interact with all the nodes in the network. This overhead may reduce the overall performance of the system. At the same time, distributed control is a benefit when each node is allowed to interact with some of the nodes only if necessary. Practically, most of the dynamic algorithms for load balancing need complete interaction between nodes of the distributed system. So, a distributed dynamic algorithm for load balancing which demands least communication between nodes is required.

In contrast to distributed methods, the non-distributed methods assign the duty of balancing the work load to a particular node or few nodes only (i.e.) not to all the nodes in the system. There are two types of non-distributed methods for dynamic load balancing. They are centralized and semi-distributed. The work proposed by L.M. Ni and Kai Hwang [12], Y. Chow and W. Kohler [40] deals with centralized non-distributed methods in which the algorithm is executed by only one node of the system (i.e.) the central node. The central node is merely responsible for load balancing of the entire system. All the other nodes have to interact with the central node only. In a semi-distributed method presented by I. Ahmed and A. Ghafoor [39], nodes of the system are divided into clusters. In each cluster, load balancing is done in a centralized manner. For each cluster, a central node is chosen. The central node in each cluster is responsible for load balancing in that cluster. With the co-operation of the central nodes in each cluster, load balancing of the entire system is accomplished. This method proposed by I. Ahmed and A. Ghafoor [39] suits for distributed systems with lot of nodes.

Since each of the nodes in the system interact only with the central node, centralized methods for dynamic load balancing needs only less number of messages to take a decision for load balancing. At the same time, centralized methods have the risk of reduced performance when the central node fails. For networks with a maximum of 100 nodes (small networks), the work done by S. Zhou [17] has exhibited that centralized load balancing is appropriate.

Modules of an algorithm for dynamic load balancing

Relying on the current status of the work load, decisions for load distribution should be taken by a dynamic load balancing algorithm. These algorithms should have two important methods. One method for gathering and maintaining the status information. Another method for helping the nodes in finding eligible jobs for load balancing. A load balancing algorithm should choose a target node to which a job is to be shifted.

Key strategies of a dynamic load balancing algorithm

The key strategies are information strategy, transfer strategy, and location strategy. Information strategy is collecting information about the nodes in the system. Transfer strategy is about choosing a job for transfer from local to a remote node. Location strategy is selecting a target node for a shifted job.

Figure-2 clearly presents the communication among modules of a dynamic load balancing algorithm. First, the transfer strategy takes the incoming jobs as input. This strategy decides whether the incoming tasks should be allocated to a remote node for balancing the load. If decision of the transfer strategy is to allocate the task, the location strategy is activated to obtain a remote node for the task. To make decisions, the information required for the transfer and location strategy is offered by the information strategy.

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

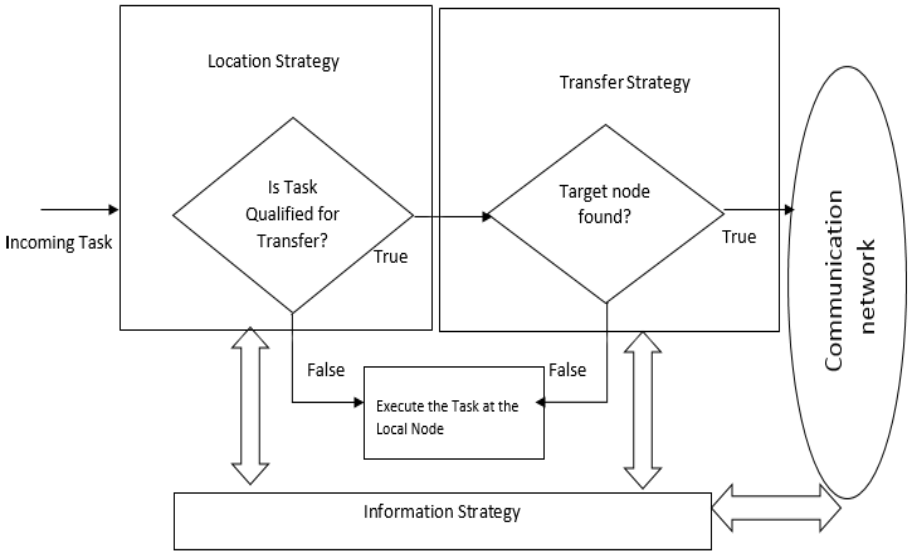

Figure-2. Communication between modules of a dynamic load balancing algorithm.

Information strategy

This strategy has the responsibility of offering information about each node in the system to the location and transfer strategies. This information helps the transfer and location strategies in taking load balancing decisions. An advanced information strategy maintains each node in the distributed system updated on the overall state of the system. As a result, additional traffic is created and an increased overhead is introduced by the algorithm. So, there is a compromise between the quantity of information exchanged and how often the information is exchanged.

The research work proposed by D.L. Eager *et al*. [6] and Y. Wang and R. Morris [20], have presented some algorithms for dynamic load balancing. These algorithms take decisions for load balancing depending on the quantity of information. The outcome of the work presented by D.L. Eager *et al*. [6] reveals those algorithms which gather large amount of information does not have a remarkable performance over algorithms which gather less amount of information.

Transfer strategy

Choosing a job for load balancing is difficult, if specific factors such as job size, job execution time, memory, and I/O are not known. This issue has been addressed by many methods.

One of the proposed methods has attempted in taking decisions for transfer of jobs regardless of job's features. This method fixes a threshold value. In this method, the job is shifted only if the size of the queue is greater than the threshold value. If not, the job is completed in the local node itself. Generality is the key benefit of this method. The disadvantage of the method is jobs of various sizes are treated differently. Algorithms of this kind for load balancing were presented in [5], [6], and [43].

A different set of methods proposed in [17], [19], [41] applies information about the current behavior of a job to evaluate the behavior of a job in the future. These methods improve the way of choosing a suitable job for load balancing. But, this works for certain workloads only. The work presented by K. Goswami *et al*. [19] chooses jobs for load balancing. This job selection is done, depending on their resource needs in the future. This need for resources was evaluated by applying a statistical method developed by M. Devarakonda and R. Iyer [42]. A technique presented by A. Svensson [41] uses execution time of jobs to distinguish small and big sized jobs. By doing so, the technique imposes the small jobs to be completed in the local node itself.

A different class of method for load balancing was presented by A. Karimi et al. [10]. An automated tool was used by this method to assess the execution time of jobs in the future. By using the trace of job's behavior under various loads, job's execution time in the future was assessed. The issue with the method is, the additional overhead introduced by tracing the job's behavior under various conditions.

There two important problems related to load balancing which relies on the transfer strategy. They are (i) correct time to start the activity and (ii) tasks chosen for the activity. Two techniques to initiate the load balancing are: arrival time of a new task at a node and departure time of the job after service from the node. Load balancing algorithms can be categorized as sender-initiated and receiver-initiated. Sender-initiated algorithms takes load balancing decisions when a new task arrives at a node. Receiver-initiated algorithms takes decisions for load balancing when the task departs after the completion. In sender-initiated methods, a node which is overloaded may request the other nodes to take its load. But in receiver-initiated methods, a node which is not heavily loaded expresses its interest to take the load of other nodes.

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

The work presented by D.L. Eager *et al.* [43] makes it clear that, choosing receiver-initiated or sender-initiated methods depends on the amount of load in the system. Sender-initiated methods are appropriate for systems which are not heavily loaded. But, receiver-initiated methods can be used for the systems with heavy work load.

Another important consideration should be the selection of qualified jobs for transfer. There are two techniques to ascertain the qualified jobs. These techniques are consider-all and consider-new-only. The consider-all technique proposed by K. Goswami *et al*. [19] ponders all the qualified jobs for load balancing. This technique is complicated than the consider-new-only. The reason is, it uses an additional method for the choosing a suitable job from a list of active jobs. The consider-new-only technique presented in [5], [6], [28], [40] takes into account the newly arrived tasks only for load balancing. This technique is widely used. As proposed by K. Goswami et al. [19], consider-all technique has shown improved results than consider-new-only technique when there is a great difference between the job size and the resource requirements.

Location strategy

Relevant to load balancing, an algorithm should correctly choose the target node to which the job should be transferred. The location strategy helps a node with overload in finding a node with a less load.

Relying on the current load at the node, a remote node is chosen. Usually, the queue length of the processor in a node estimates a node's load. A queue length is the task in service and the number of tasks waiting for service. To choose a target node for shifting the task, the quantity of information used by the location strategy is important. This paper also gains an insight into various techniques of location strategy. The widely used techniques of location strategy for choosing a target node are negotiation, random and probing.

Negotiation technique: This type of location strategy is commonly used in dynamic distributed algorithms, where each node negotiates with the other nodes in the system for taking load balancing decisions. Two negotiation-based location strategies are bidding and drafting. Bidding technique was proposed by J. A. Stankovic and I. S. Sidhu [14] and drafting technique was presented by L.M. Ni *et al*. [30].

The bidding technique makes a node with heavy load to start the load balancing. This technique is related with sender-initiated algorithms. An overloaded node broadcasts request-for-bid message to the nodes in the network. The request-for-bid message comprises information about the current load of the origin node and the tasks to be shifted. After the message is received, a target node equates the information in the message with its current status. If the load of the target node is lesser than the load of the origin node, the target node responds with a bid-message. If not, the target node will not make any response. The content of a bid message encompasses the current load of the target node and the extra load this node could handle. On receiving the bid messages from all the nodes, the origin node chooses the target node with the best bid. Best bid is determined by the node with the least load. The key issue with bidding technique is a node with least load may be overloaded. To overcome this issue, a limit on the count of accepted bids could be enforced.

Contrast to bidding technique, the drafting technique makes node with a light load to start load balancing. This drafting technique is associated with receiver-initiated algorithms. Based on the current load, the drafting technique groups nodes into three classes. They are H-load (heavily loaded), N-load (neutrally loaded) or L-load (lightly loaded).

Each of the nodes in the system maintains a status table. This table is periodically updated by all the nodes in the system with the changes in their load.Each node checks its state periodically. If a node is in L-load, it finds the H-load nodes and transmits a draft-request message. By sending such messages, an L-load node shows its readiness to receive more load. On receiving the draft-request message, a target node responds by transmitting a draft-response message if it is in the H-load state.

The content of a draft-response message is, the qualified jobs for transfer at the target node (drafted). If the original node has received many draft-response messages or in case of time-out, a target node is chosen relying on a specific criteria. Then, the information is conveyed to the drafted node by means of draft-select message. If the drafted node is still in H-load state, the drafted node shifts some tasks to the origin node. On comparison of bidding and drafting techniques in a similar setup, the outcome of the methods presented in L.M. Ni et al. [30] shows that the drafting technique outclasses the bidding technique.

Random technique: The method proposed by D.L. Eager et al. [6] and [43] shows that, a target node is chosen randomly by a node whose task is to be transferred. The target node executes the received task, based on a condition that its work load is less than the threshold defined for the queue length. If the condition is not satisfied, the target node will choose a new target node for this task. To overcome the problem of having the task passed between nodes, a maximum limit is set for the number of hops. When the number of hops reaches this limit, the node finally receives the task should execute the task irrespective of its work load. The work presented by D.L. Eager et al. [6] reveals that this technique of location strategy has shown enhanced results when compared to systems without any load balancing methods.

Probing technique: This technique of location strategy finds a target node, by making the local node to choose a subset of nodes randomly and polls each of the nodes in the subset to identify an appropriate target node. The identified node will offer service to the shifted task with an improved response time. The location strategies based on probing technique were proposed by D.L. Eager *et al.* [6] and S. Chowdhury [5]. The techniques proposed

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

by D.L. Eager *et a*l. [6] relied on threshold value and shortest queue length. The method developed by S. Chowdhury [5] was based on greedy technique.

In the work proposed by D.L. Eager *et al*. [6], a technique based on threshold was developed. A probe is done by the local node on the randomly chosen target node, which checks whether shifting a task to the target node does not affect the threshold of the target node. If the local node finds that the threshold of the target node is not affected, the task is shifted to the target node. If the threshold is affected, a new target node should be chosen at random and it has to be probed. This algorithm fixes a limit on the number of times the probing should be done by the local node. If the limit is reached, the task should be executed by the local node itself.

The shortest location strategy technique developed by D.L. Eager *et al.* [6] randomly chooses a subset of target nodes and examines the members of the subset to get their current work load. The target node with the minimum queue length (i.e) work load is chosen. If this node's work load is less than the threshold, the task is shifted to the target node. If not, the task is executed in the local node. From the results of the approach presented by D.L. Eager *et al*. [6], it is clear that the shortest location strategy is wiser than the threshold in choosing target nodes. To choose a target node, the greedy location strategy offered by S. Chowdhury [5] examines the nodes in a cyclic manner. On comparing the threshold and greedy probing techniques of location strategy, greedy technique has not shown better performance than the threshold technique.

Important factors

Algorithms developed for dynamic load balancing includes a lot of factors. Stability of the system, load measurement and performance evaluation are the key factors to be considered. This section presents these important factors.

Stability of the system: An algorithm for dynamic load balancing should preserve the stability of the system. A stable algorithm for load balancing should possess the following features: (1) as presented by R. M. Bryant and R. A. Finkel [4] and D.L. Eager *et al.* [6], nodes in the system should not enter a thrashing state; (2) any two nodes in the system does not vary by more than x% of their load (3) as mentioned by A. Goscinski [8] and T. L. Casavant [22], response time should not exceed the specific limit.

Load measurement: An algorithm for dynamic load balancing algorithm takes decisions based on the current work load. Load descriptor is a key parameter applied by a load balancing algorithm. This load descriptor defines the current load in each node. Few load descriptors are context switch rate, queue length of CPU, percentage of idle time, utilization of CPU, quantity of unfinished work and resource requirements for each job.

Queue length of CPU offers a good assessment of job response time. Most of the algorithms for dynamic load balancing applies the load descriptor. The work proposed by K. Goswami *et al*. [19] illustrates that, resource requirement is a good load descriptor. Because, tasks can be serviced in a correct sequence when the resource requirement is foreseen.

Performance evaluation: The main goal of an algorithm is enhancing the performance of the system. The algorithms for load balancing should assume an index for performance. By this way, the performance improvement can be evaluated.

The work proposed in A. Goscinski [8] reveals that, an index for performance could be user-oriented, system-oriented or both. Some of system-oriented indices are utilization of resources and throughput. User-oriented indices are mean execution time of the task and mean response time. Some of the performance indices such as job wait ratio, standard deviation of a job wait time and job mean wait time could be applied to produce the performance as per the expectations.

CONCLUSIONS

This paper explores important factors pertinent to the development of algorithms for dynamic load balancing. Important factors such as choosing target nodes, choosing tasks for shifting, performance indices, load estimation, amount of information exchanged among nodes, load levels comparison, stability of the system and estimation of resource requirements are studied. The ultimate aim of this paper is gain an insight into significant factors that should be focused for developing an efficient algorithm for dynamic load balancing. By studying various algorithms for dynamic load balancing, this paper has identified some of the key problems viz. passing the task between nodes without a limit on the number of hops, additional overhead introduced by tracing the job's behavior under various conditions for estimating the execution time of jobs. By bridging the identified gaps, an efficient algorithm for dynamic load balancing can be developed.

REFERENCES

[1] Enslow Jr. P. 1978. What is a." Distributed" Data Processing System? Computer. 11(1): 13-21.

[2] Z. Khan, R. Singh, J. Alam, R. Kumar. 2010. Performance Analysis of Dynamic Load Balancing Techniques for Parallel and Distributed Systems. International Journal of Computer and Network Security. 2(2).

[3] Xueyan Tang, Samuel T. Chanson. 2000. Optimizing Static Job Scheduling in a Network of Heterogeneous Computers. In: Proceedings of the Intl. Conf. on Parallel Processing. pp. 373-382.

[4] Raymond M. Bryant et al, 1981. A Stable Distributed Scheduling Algorithm. In: Proceedings of the 2nd Int. Conf. Dist. Comp. pp. 314-323.

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

[5] Chowdhury. S. 1990. The Greedy Load Sharing Algorithms. Journal of Parallel and Distributed Computing. 9: 93-99.

[6] Derek L. Eager, Edward D. Lazowska, John Zahorjan. 1986. Adaptive Load Sharing in Homogeneous Distributed Systems. IEEE Trans. Software Eng. SE-12(5): 662-675.

[7] Efe. K. 1882. Heuristic Models of Task Assignment Scheduling in Distributed Systems. Computer. 15(6): 50-56.

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

[9] Miron Livnyand, Myron Melman. 1982. Load Balancing in Homogeneous Broadcast Distributed Systems. In: Proceedings of the ACM Comput. Network Performance Symp.) pp. 47-55.

[10] Karimi, A. F. Zarafshan, A. b. Jantan, A. R. Ramli, M. I. Saripan. 2009. A New Fuzzy Approach for Dynamic Load Balancing Algorithm. International Journal of Computer Science and Information Security. 6(1): 001-005.

[11] R. Mirchandaney, D. Towsley, J. Stankovic. 1990. Adaptive Load Sharing in Heterogeneous Distributed Systems. Journal of Parallel and Distributed Computing. 9: 331-346.

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

[13] N.G. Shivaratri, P. Krueger, M. Singhal. 1992. Load Distributing for Locally Distributed Systems. Computer. 25(12): 33-44.

[14] J. A. Stankovic, I. S. Sidhu. 1984. An Adaptive Bidding Algorithm for Processes, Cluster and Distributed Groups. In: Proceedings 4th Int. Conf. Distributed Compu. Systems. pp. 49-59.

[15] J. Stankovic. 1984. Simulations of Three Adaptive, Decentralized Controlled, Task Scheduling Algorithms. Computer Networks. 8(3): 199-217.

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

[17] S. Zhou. 1988. A Trace-Driven Simulation Study of Dynamic Load Balancing. IEEE Transactions on Software Engineering. SE-14(9): 1327-1341.

[18] A. Barak, A. Shiloh. 1985. A Distributed Load-balancing Policy for a Multicomputer. Software-Practice and Experience. 15(9): 901-913.

[19] K. Goswami, M. Devarakonda, R. Iyer. 1993. Prediction-Based Dynamic Load-Sharing Heuristics. IEEE Transactions on Parallel and Distributed Systems. 4(6): 638-648.

[20] Y. Wang, R. Morris. 1985. Load Sharing in Distributed Systems. IEEE Trans. Comput. C-34(3): 204-217.

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

[22] T. L. Casavant. 1988. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. IEEE Trans. Software Eng. 14(2): 141-154.

[23] C. Kim, H. Kameda. 1992. An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems. IEEE Trans. Comput. 41(3): 381-384.

[24] H. Stone. 1977. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. IEEE Transactions on Software Engineering, SE-3(1): 85-93.

[25] A. N. Tantawi, D. Tawsley. 1985. Optimal Static Load Balancing in Distributed Computer Systems. J. of Assoc. Comput. 32(2): 445-465.

[26] B. Blake. 1992. Assignment of Independent Tasks to Minimize Completion Time. Software-Practice and Experience. 22(9): 723-734.

[27] S. H. Bokhari. 1979. Dual Processor Scheduling with Dynamic Reassignment. IEEE Trans. Software Eng. SE-5(4): 341-439.

[28] D. Evans, W. Butt. 1993. Dynamic Load Balancing Using Task-Transfer Probabilities. Parallel Computing. (19): 897-916.

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

[30] L.M. Ni, C. Xu, T. Gendreau. 1985. A Distributed Drafting Algorithm for Load Balancing. IEEE Transactions on Software Engineering. SE-11(10): 1153-1161.

[31] J. Stankovic.1985. Bayesian Decision Theory and Its Application to Decentralized Control of Task Scheduling. IEEE Transactions on Computers. C-34(2): 117-130.

[32] S. Penmasta, A. T. Chronopoulos. 2007. Dynamic Multi-User Load Balancing in Distributed Systems. 2007 IEEE International Parallel and Distributed

©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

Processing Symposium pp. 1-10, Long Beach, CA, USA. 2007.

[33] L. M. Campos, I. Scherson. 2000. Rate of Change Load Balancing in Distributed and Parallel Systems. Parallel Computing. 26(9): 1213-1230.

[34] C.C. Hui, S. T. Chanson. 1999. Improved Strategies for Dynamic Load Balancing. IEEE Concurrency. 7(3): 58-67.

[35] A. Corradi, L. Lenoardi, F. Zamboelli. 1999. Diffusive Load Balancing Policies for Dynamic Applications. IEEE Concurrency. 7(1): 22-31.

[36] S. Dhakal, M. M. Hayat, J.E. Pezoa, C. Yang, D. Bader. 2007. Dynamic Load Balancing in Distributed System in the Presence of Delays: A Regeneration-Theory Approach. IEEE Transactions on Parallel and Distributed Systems. 18(4).

[37] D. Grosu, A. T. Chronopoulos. 2005. Noncooperative Load Balancing in Distributed Systems. Journal of Parallel and Distributed Computing. 65(9): 1022-1034.

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

[39] I. Ahmed, A. Ghafoor. 1991. Semi-Distributed Load Balancing for Massively Parallel Multicomputers. IEEE Trans. Software Eng. 17(10): 987-1004.

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

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

[42] M. Devarakonda, R. Iyer. 1989. Predictability of Process Resource Usage: A measurement-Based Study on Unix. IEEE Transactions on Software Engineering. 15(12): 1579-1586.

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