A STUDY OF LOAD DISTRIBUTION ALGORITHMS IN DISTRIBUTED SCHEDULING

Shounak Chakraborty[1], Ajoy Kumar Khan[2]

[1]*Department of Information Technology, Assam University, India*
[2]*Department of Information Technology, Assam University, India*

Abstract

As we know that in distributed systems several autonomous computers are interconnected to provide a single coherent view of a powerful system and these autonomous computers work independently in a team-like fashion such that the domain of tasks is shared between all of them. Now to take full advantage of this distributed scenario, we need good resource allocation schemes. Load distribution algorithm's work is to deliverately distributed and re-distributes the tasks (loads) among all the participating nodes so that the overall performance of the entire system is maximized. in this paper we study the details of the load distribution algorithms and their suitableness in various load scenerios.

Keywords: *loads, stability, Load balancing, load sharing, location affinity*

----------***----------

1. INTRODUCTION

A distributed scheduler is a resource management component of a distributed operating system that focuses on judiciously and transparently redistributing the load of the system among the individual units to enhance overall performance. Users submit tasks at their host computers for processing. The need for load distribution arises in such environments because, due to the random arrival of tasks and their random CPU service time requirements, there is a good possibility that several computers are idle or lightly loaded and some others are heavily loaded, which would degrade the performance. In real life applications there is always a possibility that one server or system is idle while a task is being waited upon at another server. [1]

Let us consider a system of N independent servers and let ρ be the utilization of each server. Then P=1-ρ is the probability of a server being idle. The probability P is given by the expression [1] as following:

$$P= \sum_{i=1}^{N} \binom{N}{i} Q_i H_{N-i} \qquad \text{(Eq. 1.1)}$$

Where Q_i is the probability that a given set of i server are idle and H_{N-i} is the probability that a given set of (N-i) servers is not idle.

Even in a homogeneous distributed system, system performance can potentially be improved by transferring the load from the heavily loaded systems (sender) to lightly loaded systems (receiver). The performance is most often quantified by average response time of tasks (time interval between submission and completion of task) in case of distributed systems. Moreover, resource queue length and CPU queue length are good indicators of load since they co-relate well with the task response time. [2]

2. COMPONENTS OF A LOAD DISTRIBUTION ALGORITHM

2.1 Transfer Policy

Transfer policy indicates when a node (system) is in a suitable state to participate in a task transfer. The most popular proposed concept for transfer policy is based on a optimum threshold. [1], [3], [4], [5]. Thresholds are nothing but units of load. When a load or task originates in a particular node and the number of load goes beyond the threshold T, the node becomes a sender (i.e. the node is overloaded and has additional task(s) that should be transferred to another node). Similarly, when the loads at a particular node falls bellow T it becomes a receiver.

2.2 Selection Policy

A selection policy determines which task in the node (selected by the transfer policy), should be transferred. If the selection policy fails to find a suitable task in the node, then the transfer procedure is stopped until the transfer policy indicates that the site is again a sender. Here there are two approaches viz.: pre-emptive and non-pre-emptive. Non-pre-emptive the approach is simple, we select the newly originated task that has caused the node to be a sender, for migration. But often this is not the best approach as the overhead incurred in the transfer of task should be compensated for by the reduction in the response time realised by the task. Also there are some other factors, firstly the overhead incurred by the transfer should be minimal (a task of small size carries less overhead) and secondly, the number of location dependent system calls made by the selected task

should be minimal. This phenomenon of location dependency is called location affinity and must be executed at the node where the task originated because they use resources such as windows, or mouse that only exist at the node.

2.3 Location Policy

A location policy decides to which node a task selected for transfer should be transferred. A widely used method by the researchers is polling, [1],[3],[4],[5],[6], which involves querying another node whether it can accept a task to be transferred to it. The variations are:

- **Random:** It is a simple dynamic technique where a task is simply transferred to a random node. Obviously there is a limit to the number of transfers.
- **Threshold:** Here polling is done to the node before task transfer to check its status.
- **Shortest:** This technique selects the most lightly loaded node after polling the nodes at random.

2.4 Information Policy

This is responsible for deciding when information about the other nodes in the system should be collected, where it should be collected from and what information should be collected.

The variants of this scheme are demand driven, periodic and state change driven.

2.5 Stability

We first informally describe two views of stability: the queuing theoretic perspective and the algorithmic perspective.

2.5.1 Queuing Theoretic Perspective

According to the queuing theoretic perspective, when the long-term arrival rate of work to a system is greater than the rate at which the system can perform work, the CPU queues grow without bound. Such a system is termed unstable.[7]

For example we consider a load distribution algorithm performing excessive message exchanges to collect state information. The sum of the load due to external work arriving and the load due to the overhead imposed by the algorithm can become higher than the service capacity of the system, causing instability.

We use the effectiveness of an algorithm as the evaluating criterion. A load distributing algorithm is said to be effective under a given set of conditions if it improves the performance relative to that of a system not using load distributing.

2.5.2 The Algorithmic Perspective

If an algorithm can perform fruitless action with finite probability, the algorithm is said to be unstable. [6] We consider processor thrashing, the situation where the transfer of a task to a receiver may increase the receiver's queue length to a point of overload, necessitating the transfer of that task to yet another node. If this process repeats indefinitely then according to [6] it is unstable.

3. TYPES OF LOAD DISTRIBUTION ALGORITHM

Load distribution algorithms can be categorized according to the taxonomy introduced by Casavant and Kuhl in [9].

3.1 Classification According to Approach

Load distribution algorithms can be classified as static, dynamic or adaptive. Static schemes are those when the algorithm uses some priori information of the system based on which the load is distributed from one server to another. The disadvantage of this approach is that it cannot exploit the short term fluctuations in the system state to improve performance. This is because static algorithms do not collect the state information of the system. These algorithms are essentially graph theory driven or based on some mathematical programming aimed to find a optimal schedule, which has a minimum cost function. But unfortunately Gursky has shown that the proble of finding an optimal schedule for four or more processing elements is NP-hard.

Dynamic scheduling collect system state information and make scheduling decisions on these state information. An extra overhead of collecting and storing system state information is needed but they yield better performance than static ones. Dynamic load distribution for homogenous systems was studied by Livny and Melman in [1], and the scenario of task waiting in one server and other server being idle was regarded as "wait and idle" (WI) condition. Significantly for a distributed system of 20 nodes and having a system load between 0.33 and 0.89, the probability of WI state is greater than 0.9. Thus, at typical system loads there is always a potential for improvement in performance, even when nodes and process arrival rates are homogeneous. Load sharing facility for large, heterogeneous system is studied in Utopia [8].

Adaptive load balancing algorithms are a special class of dynamic load distribution algorithms, in that they adapt their activities by dynamically changing the parameters of the algorithm to suit the changing system state.

3.2 Pre-emptive and Non pre-emptive Type

A pre-emptive transfer involves transfer of task which are partially executed. These transfers are expensive because the state of the tasks also needs to be transferred to the new location.

Non pre-emptive transfers involves transfer of tasks which has not been started. For a system that experiences wide fluctuations in load and has a high cost for the migration of partly executed tasks, non pre-emptive transfers are appropriate.[7]

3.3 Load Sharing and Load Balancing

Although both type of algorithms strive to reduce the likelihood of unshared state i.e. wait and idle state, load balancing goes a step further by attempting to equalize loads at all computers. Because a load balancing algorithm involves more task transfers than load sharing algorithms, the higher overhead incurred by load balancing types may outweigh its potential improvement.

3.4 Initiation Based

In general the algorithms are also categorized on which node initiates the load distribution activity. The variations are sender initiated, receiver initiated or symmetrically initiated (by both sender and receiver).

A sender initiated algorithm was studied by Eager et. al. in [4] and a receiver initiated algorithm was studied in [3], where as a symmetrically initiated algorithm was adopted in [10]. Moreover an adaptive stable symmetrically initiated algorithm was put forward in [5] and a stable sender initiated algorithm was discussed in [7].

All the load distribution algorithms are based on one of more of the types discussed above.

4. COMPARISON AND CONCLUSION

The observations on various kinds of algorithms are as follows:

- Sender initiated algorithms work well in low system load, but in case of high system load when most of the nodes are senders they send query to each other resulting in wastage of CPU cycles and incurring more delay due to which the system becomes unstable.
- This un-stability happens with receiver initiated algorithms when the system load is low and most nodes are receiver.
- For symmetrically initiated algorithms, they cannot use the previous gathered information and so in stateless.
- Adaptive algorithms use the previous information to query a new node and also adjust their threshold themselves according to the information.

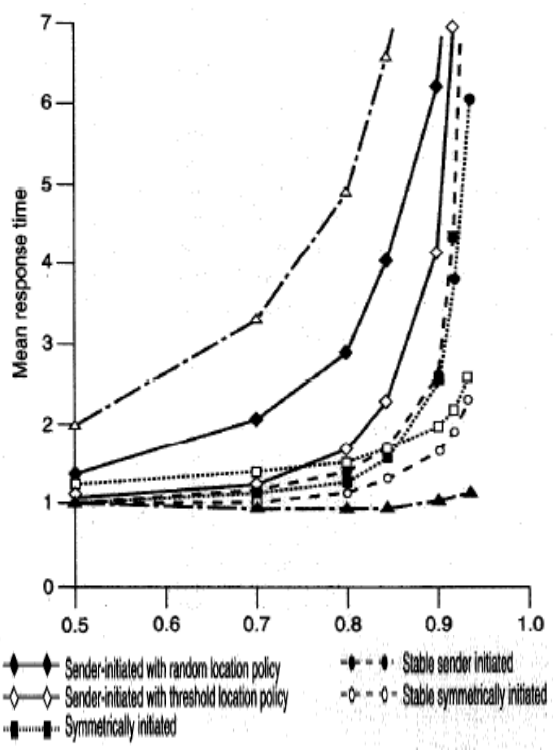

Fig 1: Average response time versus system loads [7]

Based on the performance trends of load sharing algorithms, the recommendations [11], for selection of a load distribution algorithm are:

- If the system under consideration never attains high load, sender-initiated algorithms will give an improved average response time over no load sharing at all.
- Stable scheduling algorithms are recommended for systems that can reach high loads. These algorithms perform better than non-adaptive algorithms for the following reasons:
    a) For overloaded processors are even more hurled with the extra task of message handling, in case of sender initiated algorithms
    b) And for the receiver initiated algorithm, which works well on high loads, but the pre-emptive transfers are expensive.
- For a system that experiences a wide range of load fluctuations, the stable symmetrically initiated scheduling is recommended because it provides improved performance and stability over entire spectrum of system loads.
- For a system that experiences wide fluctuations in loads but has a high cost for the pre-emptive transfers, a stable sender initiated is recommended.
- For a system that experiences heterogeneous work arrival, adaptive stable algorithms are preferred.

REFERENCES

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

[2]. Zhou S., "An Experimental Assesment of Resource Queue Lengths as Load Indices" *Proc. of USENIX*, Washington, pp 73-82, 1987.

[3]. D.L. Eager, E.D. Lazowska. and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation*, Vol. 6, No. 1, pp. 53-68, 1986.

[4]. D.L. Eager, E.D. Lazowska, and J. Zahorian, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Eng.*, Vol. 12. No. 5, pp. 662-675, 1986

[5]. N.G. Shivaratri and P. Krueger. "Two Adaptive Location Policies for Global Scheduling," Proc. 10th *Int'l Conf. Distributed Computing Systems. IEEE COMPUTERS*, pp. 502-509, 1990

[6]. Bryant R.M. and R.A. Finkel. "A Stable Distributed Scheduling Algorithm." *Proc. 2nd Int'l Conference On Distributed Computing*, pp 314-323, 1981

[7]. N.G. Shivaratri, P.Krueger and M.Singhal. "Load Distributing for Locally Distributed Systems". *Proc. of IEEE Computers*, 1992.

[8]. Zhou S., Z.Zheng, J.Wang and P.Delisle, "Utopia: A Load Sharing Facility for Large Heterogeneous Distributed Computer Systems". University of Toronto Press, 1992.

[9]. Casavant T.L. and J.G. Kuhl. "A taxonomy of Scheduling in General-Purpose Distributed Computing Systems", *IEEE Transactions on Software Engg.*, pp. 141-154, 1988

[10]. P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies." *Proc. 7th Int'l Con& Distributed Computing Systems, IEEE CS*, pp. 242-249, 1987.

[11]. N.G. Shivaratri and M. Singhal. "Advanced Concepts In Operating Systems". Tata McGraw Hill Edu. Pvt. Ltd., ISBN: 13:978-0- 07-047268-6, 30th edition, 2012.