Dynamic Load Balancing Algorithm Complexity

Sung D. Cha

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Technical Report 87-24

Dynamic Load Balancing Algorithm Complexity

Sung D. Cha

Department of Information and Computer Science
University of California, Irvine
Irvine, California 92717

Abstract

This paper presents a theoretical analysis of the asymptotic complexity inherent in a load balancing algorithm in a loosely-coupled network, where processor communication is achieved by message passing. The load balancing complexity depends on the network topology and the overhead of processor communication for each polling strategy. The best, worst, and average case analysis of the load balancing algorithms for the various polling topologies are presented. The polling strategies considered are local, global, and random polling. The complexity is presented as a function of the number of processors in the network.

1 Introduction

A major advantage of a distributed system is the possibility of achieving load balancing and fault tolerance. When the current load of a processor exceeds its processing capability, other underloaded processors, if there are any, might be able to share the load to better utilize the processors and to maximize the system throughput. Furthermore, if there exist multiple paths between any pair of processors, a limited degree of fault tolerance against processor and/or channel failures can be achieved. The processors could communicate despite communication channel and/or intermediate processor failures. The degree of fault tolerance that can be achieved depends on the degree of processor and/or channel redundancy provided in the system – e.g., the system topology. Therefore, careful analysis is needed to achieve the maximum fault tolerance with the minimum cost possible.

A distributed system considered in this paper has a point-to-point interconnection structure [LPS81], where each processor can communicate by explicit message passing through the communication channels. If two processors are not directly connected, the message can be forwarded by the intermediate processors.

This paper examines the complexity of a typical fully distributed and asynchronous dynamic load balancing algorithm. The algorithm is distributed in that there is no central processor that controls the load balancing activity, and any processor can initiate the load balancing activity independent of the other processors. The algorithm is asynchronous in that the processors do not need to be synchronized to achieve load balancing. Once the source and destination processor are determined, the load balancing activity can take place independent of other processors. The algorithm is dynamic in that the processor selected for job execution can vary dynamically depending on the each host's load when the load balancing activity is initiated. The main focus of this paper is to examine the asymptotic complexity inherent in a dynamic load balancing algorithm in a loosely-coupled network, as a function of the system topology and the number of the processors in the system.

The load balancing activity can be initiated by either the idle processors or by the overloaded processors. The former is called a receiver-initiated algorithm while the latter is called a sender-initiated one. The idle processors could "demand" more loads to better utilize the processing power. This approach, however, requires careful design because it could lead to unnecessary invocation of load balancing algorithms if the system overall is underloaded. Each processor could query other processors demanding more jobs which are not available. Load balancing algorithms based on this approach have been published by [NH85,LK87] among others. The load balancing activity, however, could be initiated by the overloaded processors seeking idle processors to share the loads. Since the load balancing activity itself is executed as a processor, the load on the already overloaded processor will be increased. However, this situation can be avoided by the careful definition of "overloading". For example, if the load at a processor exceeds a certain limit, say 98% of the processing power, the processor could be defined "overloaded" and initiate the load balancing activity. The processor could then be logically overloaded, but not physically overloaded. The parameter can be carefully adjusted by the system designers or maintainers depending on the overhead incurred by the load balancing activity.

If no idle processor can be found when the load balancing is attempted, the system is called saturated. If a system saturation occurs, load balancing cannot be achieved and other measures need to be considered. For example, the system could reject any other new job submission or terminate the least critical jobs until the system saturation disappears.

Section 2 reviews previously proposed dynamic load balancing algorithms. A distributed asynchronous dynamic load balancing algorithm is proposed in section 3. Since the algorithm is fully distributed, the algorithm requires that the processor initiating the load balancing activity poll the relevant processors to provide their current load information. The polling overhead, however, depends on the system topology and the scope of polling desired. Section 4 provides an analysis of the effect of system topology on the polling complexity, followed by the conclusions.

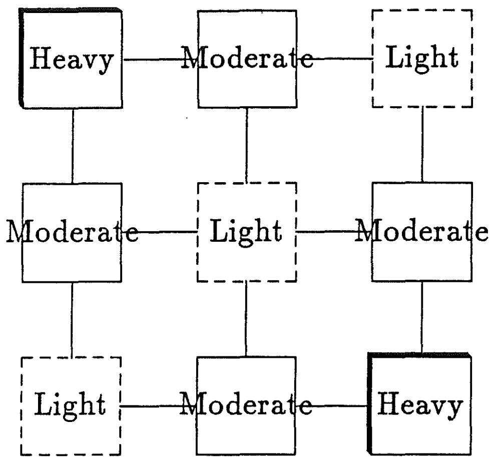
2 Related Research

The problem of load balancing in a distributed system has been a topic of much research in the past. The load balancing problem also has been called either load sharing or processor placement problem in the literature. In general, it is a part of global scheduling problem to achieve the optimal system performance. A comprehensive survey of the subject was provided by Wang and Morris[WM85].

When a job is submitted in a distributed system environment, it will be executed by one of the processors that has the resources the job requires. This initial scheduling of the job submitted is called the processor placement problem. This strategy can be viewed as a load balancing algorithm because the processor can be selected so as to balance the load among all the processors. A strategy of choosing the processor with the lightest node is used in the Purdue Engineering and Computer Network [HCG*82], where 9 DEC computers are interconnected. The processor placement problem usually assumes that job scheduling is irrevocable in that the job is executed on the processor selected and no further job migration is allowed. The load balancing algorithms, however, may allow the jobs in the queue to migrate. The tradeoff is between the possibility of improving the system throughput by allowing any waiting jobs to migrate and the increased complexity and overhead of the load balancing algorithm. The processor placement problem and the load balancing problem are equivalent if no migration of the process is allowed after the initial job scheduling. The possibility of allowing a job to migrate during the execution is not considered since the overhead of moving running processes generally outweighs the potential benefits.

The existing load balancing algorithms can be classified by the scope of balancing and the adaptiveness to the system state as follows: local vs global and static vs dynamic load balancing algorithms.

In a local load balancing algorithm, only the processors directly connected to the initiating processor are considered. In a global load balancing algorithm, all the processors in a distributed system network must be considered. Although the successive application of local load balancing activities usually leads to global load balancing, it is not necessarily the case. For example, Lin and Keller [LK87] propose a dynamic load balancing algorithm where the load is classified as light, moderate, or heavy. The load is transferred from the heavily loaded processor to the lightly loaded one if the two processors are directly connected. No load balancing activity occurs at the moderately loaded processors. If all the immediate neighbors of heavily or lightly loaded processors are moderately loaded, no load balancing activities take place as shown below.



Global load balancing algorithms, on the other hand, require increased overhead because all the processors in the network have to be consulted. In an effort to overcome the limitation of the local load balancing algorithms, random load balancing algorithms have been proposed in the literature, where a subset of processors in the network are selected randomly for consultation on their load regardless of their distance to the processor initiating the load balancing activity.

Dynamic load balancing policy takes the current system state into consideration so as to react to changes in load. Static algorithms, on the other hand, make load balancing decisions independent of the current system state. While static policies are simpler to implement and analyze, they may not be as effective as the dynamic ones. The performance of a load balancing algorithm is usually measured by the average job turnaround time. Several approaches to design and analyze dynamic load balancing algorithms have been suggested. Wang and Morris[WM85] define several classes of dynamic load balancing algorithms: diffusion, contract bidding, and state feedback.

In a diffusion approach, the load balancing is achieved by having the neighboring nodes communicate and cooperate so that the load from the overloaded processor is shifted to the underloaded processors. Either a local or a global load balancing algorithm can be implemented. The diffusion approach is general enough to include the "dipstick" approach where the job is executed locally unless the load does not exceed a prespecified limit. The processor welcomes the load from other overloaded processors if the current load is below a prespecified minimum. An algorithm proposed by Lin and Keller [LK87] is an example of the diffusion approach. The effectiveness of these load balancing algorithms depend on the communication overhead, job transfer overhead, and the careful selection of the boundary load where the load balancing activity occurs. The communication overhead in the diffusion approach is analyzed in section 4.

The bidding approach can be used in a workstation network environment where there are a number of servers and clients attached to the network. Both sender-initiated and receiver-initiated algorithms can be implemented in a bidding approach. In a sender-initiated approach, the sources broadcast the job and collect the bid from the servers. The server with the lowest bid (e.g., most lightly loaded) runs the job. An idle server could send a "request for bids", indicating its willingness to take additional jobs to implement the receiver-initiated load balancing algorithm. The probabilistic load balancing algorithm can also be classified as a bidding approach. When a job is to be executed remotely, a remote processor will be selected with some probability or at random. The selected processor can be consulted if its current load is low enough to accept the job. Examples of the bidding or probabilistic approach include the algorithms proposed by Hsu and Liu[HL86], and Eager, Lazowska, and Zohorjan [ELZ86]. It is interesting to note that a simulation study[ELZ86] shows that simple algorithm such as random host selection are about equally effective as complicated algorithms.

In a state feedback approach, the current load on each host in the system is collected and updated periodically. Any new job submitted is routed to the most lightly loaded processor, as used in Purdue ECN. In a loosely-coupled network environment, the overhead of periodically updating the load information on each host can be significant. If the load information is updated less frequently, the possibility of poor judgement due to the out-of-date load information could result in a reduced system output. An examples of the state feedback approach is the algorithm proposed by Hać and Johnson[HJ86]. A token circulates the network periodically, collecting and updating the load information on each host. Whenever a new job is submitted, the host with the lightest load is selected as an execution site.

Another approach is a heuristic approach suggested by Elf[Efe82]. Given the set of jobs and the interprocess communication cost among the jobs, the module cluster algorithm forms clusters to which a processor can be assigned so as to minimize the communication cost among the clusters. If load imbalancing results, the module reassignment algorithm shifts the jobs until the load is balanced.

3 A Dynamic Load Balancing Algorithm

A proposed algorithm assumes that the following information is either available or can be computed:

- A distributed system consists of N processors, denoted by PE_i, whose processing power is denoted by p_i $(1 \leq i \leq N)$.

- The average load of processor i at any time is denoted by l_i $(1 \leq i \leq N)$. Thus, a processor is overloaded if $l_i > p_i$. The load balancing algorithm is independent of the algorithm that computes the load.

- The minimum distance between any two processors is defined by the N by N matrix MD, where MD_{ij} is the minimum distance between the processors i and j. Distance from any processor to itself is 0. The distance does not have to be the physical distance between the processors. Other measures such as communication and job transfer overhead or the number of intermediate processors on the path can be used.

A typical dynamic load balancing algorithm is given below:

```
(1) loop
(2)    the initiating processor, say $PE_i$, polls relevant neighbors on current load ;
(3)    $PE_i$ finds an idle processor $PE_j$ such that $(p_j - l_j) - MD_{ij}$ is positive and maximum;
(4)    if such $PE_j$ is found
(5)       then transfer $p_j - l_j$ amount of load from $PE_i$ to $PE_j$
(6)       else the system is saturated
(7) end loop;
```

The processor initiating the load balancing activity depends on whether sender-initiated or receiver-initiated algorithms are used. The initiating processor must poll other processors on their current load information. The scope of polling depends on whether global or local load balancing policies are adapted. The load balancing activity, shown in lines (2) through (5), chooses the target processor where the net benefit of load balancing (reduced load − transfer overhead) is maximized.

The complexity of the above algorithm shown in line (3) and (4) is $O(N)$, where N is the number of processors in the system. The complexity of line (2) depends on the distributed system topology and whether local or global load balancing is desired. If local load balancing is desired, the polling of the current load of each processor is limited only to the immediate neighbors, whereas global balancing requires all processors in the distributed system to be polled. The relationship between system topology and polling complexity is the subject of the next section.

4 Polling Complexity and System Topology

This section investigates the complexity of polling the relevant processors over the network for various system topologies. The analysis consists of best, worst, and average case for both local and global polling. The complexity of random polling is also analyzed. This knowledge can be used by the distributed system designer to choose the topology to minimize the polling overhead.

The overhead of polling is measured as the number of messages required for communication among the relevant processors. In the case of local polling where only the immediate neighbors are consulted, this is equivalent to the number of messages that the initiating processors must generate. Global load balancing, on the other hand, may require some intermediate processors to store and forward the polling messages if no direct link is available. The number of times the messages are stored and forwarded before reaching the destination is measured as the global polling overhead. The study conducted by Lazouska and others[LZCZ84] shows that the processor cost of packaging data for transmission, routing cost by the intermediate processor, and unpackaging upon reception far outweigh the communication network costs of transmitting the data.

The distributed system topology is modeled as a connected (undirected) graph where each processor is represented as a node and any direct (duplex) communication channel between two processors is represented as an edge. The minimum number of edges in the graph is $N-1$ since N nodes must form a connected graph. The maximum number of edges in the graph is $\binom{N}{2}$ when the graph is fully connected.

4.1 Local Polling

4.1.1 Best Case

The best case is when the system topology is linear, where the number of immediate neighbors is one or two. Thus, the polling complexity is $O(1)$.

4.1.2 Worst Case

If the system is fully connected, a processor has to poll all $N-1$ neighbors. Thus, the complexity is $O(N)$.

4.1.3 Average Case

The average number of edges in a connected graph with N nodes varies from $N-1$ if the graph is linear to $\binom{N}{2}$ if graph is fully connected. If every topology is equally likely to occur, the average number of edges connected per node is given as follows:

$$\begin{aligned} E &= \frac{(\text{sum of all integers between } N-1 \text{ and } \binom{N}{2})}{N * (\text{number of integers between } N-1 \text{ and } \binom{N}{2})} \\ &= \frac{\sum_{i=1}^{\binom{N}{2}} i - \sum_{i=1}^{N-2} i}{N * (\binom{N}{2} - (N-2))} \\ &= \frac{(N-1)(N+2)}{4N} \end{aligned}$$

$$= O(N)$$

Therefore, the average local polling complexity is $O(N)$.

4.2 Global Polling

The complexity of global polling can be analyzed using the minimum distance matrix MD, where MD_{ij} is the minimum number of edges required to traverse from node i to j. Thus, the number of times the message has to be stored and forwarded before delivery is used as a complexity measure. The matrix is symmetric by definition and the values of all the diagonal elements are zero.

4.2.1 Best Case

The best case is $O(N)$ when the system is fully connected since each processor will make $N-1$ polls to its neighbors (e.g., $MD_{ij} = 1$).

4.2.2 Worst Case

The worst case occurs when the system has a linear topology. If we assume that only the destination examines the actual contents of the message in a point-to-point network, global polling requires sending $N-1$ messages, where intermediate edges on the path varies from 1 to $N-1$. Therefore, the complexity is $\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} = O(N^2)$.

4.2.3 Average Case

The average global polling complexity is same as the average value of the sum of a row or a column of the matrix. From the definition of the matrix, the following can be derived:

- The matrix MD is symmetric and the N diagonal elements have value zero.
- There are at least $N-1$ elements with value 1. That is, the graph has at least $N-1$ edges.
- The other $(N^2 - 2N + 1)$ elements have a value between 1 and $(N-1)$.

Suppose the maximum element of a row or a column is k $(1 \leq k \leq N-1)$, where k is assumed to be chosen equally likely. From the definition of the matrix MD, the following can be derived:

- The diagonal element has value 0.
- There are k elements with values 1 through k since there are exactly $k-1$ intermediate nodes on the longest path.

- The other elements $(N-k-1)$ can have any value between 1 and k. Again, it is assumed that any value between 1 and k can be chosen with equal likelihood.

The average value of the sum of a row if the maximum element is k is then

$$0+\sum_{i=1}^{k} i+(N-k-1)\frac{1}{k}\left(\sum_{i=1}^{k} i\right)$$

which is $\frac{(N-1)(k+1)}{2}$. For example, if the system is fully connected, the value of k is 1, requiring $N-1$ messages. However, if the system is linearly connected, k is $N-1$, requiring $\frac{N(N-1)}{2}$ polls. These values match the results presented in the best and worst case analysis.

Since it is assumed that the maximum value k is assumed to be chosen equally likely between 1 and $N-1$, the expected sum of the elements of a row or a column is then

$$\begin{aligned}\frac{1}{N-1}\left(\sum_{k=1}^{N-1}\frac{(N-1)(k+1)}{2}\right) &= \sum_{k=1}^{N-1}\frac{(k+1)}{2} \\ &= \frac{(N^2+N-2)}{4}\end{aligned}$$

Therefore, the average complexity of the global polling is $O(N^2)$.

4.3 Random Polling

In addition to the local or global polling strategies, a random polling strategy has been proposed in the literature [HL86,ELZ86]. In a random polling strategy, the processor initiating the load balancing activity randomly selects a subset of processors in the network to consult on their current load information. It is assumed that each processor is equally likely to be selected regardless of the its distance to the polling processor.

The complexity of random polling algorithm can be analyzed using the minimum distance matrix, MD, developed in the previous subsection. It is shown that the average complexity of global polling, which is equivalent to the average sum of a row or a column of minimum distance matrix, is $\frac{(N^2+N-2)}{4}$.

Suppose m out of the $N-1$ processors (excluding the polling processor itself) are polled randomly. The complexity of the random polling strategy is then equivalent to the average sum of arbitrary m elements of a row or a column of the minimum distance matrix, which is

$$\frac{(N^2+N-2)}{4}\,\frac{m}{N-1}$$

because each element is assumed equally likely to be selected. Therefore, if the number of processors randomly polled is proportional to the number of processors in the network, e.g., $O(N)$, the complexity of random polling strategy is $O(N^2)$. This complexity is equivalent to that of the global polling strategy. If, however, the number of polled processors is constant and independent of the network size, the random polling complexity is $O(N)$, which is equivalent to that of the local polling.

5 Conclusion

This paper proposes a new asynchronous load balancing algorithm and presents a theoretical analysis of its inherent asymptotic complexity in a distributed system with loosely-coupled network. The complexity, represented as a function of the number of processors in the network, N, depends on the network topology and the scope of load balancing (polling strategy), is shown below:

Strategy	Best Case	Average Case	Worst Case
Local Polling	$O(1)$ Linear topology	$O(N)$ Arbitrary topology	$O(N)$ Fully-connected topology
Global Polling	$O(N)$ Fully-connected topology	$O(N^2)$ Arbitrary topology	$O(N^2)$ Linear topology
Random Polling	$O(N)$ if number of processors polled is constant $O(N^2)$ if number of processors polled is proportional to N		

Therefore, it is shown that the polling overhead outweighs the target selection overhead. This analysis can be considered conservative because the cost of message store and forward by an intermediate processor is assumed to be the same as that of each comparison to select the target in the polling processor. In a real implementation, the cost of message passing is likely to be considerably greater than that of a comparison.

The complexity of random polling strategy has also been analyzed. It is shown that the random polling complexity is equivalent to that of the local polling if the number of polled processors is constant and independent of the network size. If the number of polling processors is linearly proportional to the size of the network, the polling complexity is same as that of the global polling strategy.

References

[Efe82] Kemal Efe. Heuristic models of task assignment scheduling in distributed systems. *IEEE Computer*, 50–56, June 1982.

[ELZ86] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transaction on Software Engineering*, SE-12(5):662–675, May 1986.

[HCG*82] Kai Hwang, William J. Croft, George H. Goble, Benjamin W. Wah, Faye A. Briggs, William R. Simmons, and Clarence L. Coates. A unix-based local computer network with load balancing. *IEEE Computer*, 55–66, April 1982.

[HJ86] Anna Hać and Theodore J. Johnson. A study of dynamic load balancing in distributed system. In *Proceedings of ACM SigComm Symposium on Communications, Architectures, and Protocols*, pages 348–356, August 1986.

[HL86] Chi-Yin Huang Hsu and Jane W. S. Liu. Dynamic load balancing algorithms in homogeneous distributed systems. In *International Conference on Distributed Computing Systems*, pages 216–223, 1986.

[LK87] Frank C.H. Lin and Robert M. Keller. The gradient model load balancing method. *IEEE Transaction on Software Engineering*, SE-13(1):32–38, January 1987.

[LPS81] B.W. Lampson, M. Paul, and H.J. Siegert, editors. *Distributed Systems.* Springer-Verlag, 1981.

[LZCZ84] E.D. Lazowska, J. Zahorjan, D.R. Cheriton, and W. Zwaenepoel. *File Access Performance of Diskless Workstation.* Technical Report 84-06-01, University of Washington, Seattle, June 1984.

[NH85] Lionel M. Ni and Kai Hwang. Optimal load balancing in a multiple processor system with many job classes. *IEEE Transaction on Software Engineering*, SE-11(5):491–496, May 1985.

[WM85] Yung-Terng Wang and Robert J.T. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3):204–217, March 1985.