World Academy of Science, Engineering and Technology
International Journal of Civil and Environmental Engineering
Vol:2, No:2, 2008

Performance Analysis of Load Balancing Algorithms

Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma

***Abstract*—**Load balancing is the process of improving the performance of a parallel and distributed system through a redistribution of load among the processors [1] [5]. In this paper we present the performance analysis of various load balancing algorithms based on different parameters, considering two typical load balancing approaches static and dynamic. The analysis indicates that static and dynamic both types of algorithm can have advancements as well as weaknesses over each other. Deciding type of algorithm to be implemented will be based on type of parallel applications to solve. The main purpose of this paper is to help in design of new algorithms in future by studying the behavior of various existing algorithms.

***Keywords*—**Load balancing (LB), workload, distributed systems, Static Load balancing, Dynamic Load Balancing

I. INTRODUCTION

IN parallel and distributed systems more than one processors processing parallel programs. The amount of processing time needed to execute all processes assigned to a processor is called workload of a processor. A system [2] [3] of distributed computers with tens or hundreds of computers connected by high speed networks has many advantages over a system that has the same standalone computers. A distributed system provide the resource sharing as one of its major advantages, which provide the better performance and reliability than any other traditional system in the same conditions. One of the research issues in parallel and distributed systems is the development of effective techniques for distributing workload on multiple processors. The main goal is to distribute the jobs among processors to maximize throughput, maintain stability, resource utilization and should be fault tolerant in nature. Local scheduling performed by the operating system consists of the distribution of processes to the time-slices of the processor. On the other hand Global scheduling is the process of deciding where to execute a process in a multiprocessor system. Global scheduling may be carried out by a single central or master processing element, or it may be distributed among the processing elements. Global scheduling is further classified into static and dynamic scheduling categories. In static scheduling processes are assigned to processors before the executions starts. On the other hand dynamic scheduling can reassign the processes to the processors during the execution. Load sharing and load balancing are the further classifications of dynamic scheduling. Load sharing struggle to avoid the unshared state in processors which remain idle while tasks compete for service at some other processor. Load balancing also do the same but it goes one step ahead of load sharing by attempting to equalize the loads at all processors. Load balancing is to ensure that every processor in the system does approximately the same amount of work at any point of time. Processes may migrate from one node to another even in the middle of execution to ensure equal workload. Algorithms for load balancing have to rely on the assumption that the on hand information at each node is accurate to prevent processes from being continuously circulated about the system without any progress. Load balancing is one of prerequisites to utilize the full resources of parallel and distributed systems. Load balancing may be centralized in a single processor or distributed among all the processing elements that participate in the load balancing process.

Several tasks are scheduled for separate processors, based on the current load on each CPU. Many researches have been carried out on load balancing for many years with the aim is to find the load balancing schemes with overhead as low as possible.

S. Sharma is with the Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar – 143001, Punjab, India (Phone: +91-98555-90815; e-mail: Sandeep_gndu@ yahoo.com).

S. Singh is with the Department of Computer Science and IT, DAV College, Amritsar – 143001, Punjab, India (e-mail: sarabjit_gndu@ yahoo.com).

M. Sharma is with the Department of Computer Science and Engineering, Sai Engineering College, Pathankot – 145001, Punjab, India (e-mail: meenu_ddit@yahoo.com).

II. PAPER ORGANIZATION

In our paper we have carried out the study of six load balancing algorithms, various parameters are used to check the results. In this paper first Introduction is given then in III brief introduction of static load balancing algorithms, IV gives introduction of dynamic load balancing algorithms, in V we have given parameters we selected to analyze algorithms VI gives the study of results with the help of table I and conclusion is given in VII.

III. STATIC LOAD BALANCING

In this method the performance [3] [6] of the processors is determined at the beginning of execution. Then depending upon their performance the work load is distributed in the start by the master processor. The slave processors calculate their allocated work and submit their result to the master. A task is always executed on the processor to which it is assigned that is static load balancing methods are non-preemptive. The goal of static load balancing method is to reduce the overall execution time of a concurrent program while minimizing the communication delays. A general disadvantage of all static schemes is that the final selection of a host for process allocation is made when the process is created and cannot be changed during process execution to make changes in the system load.

A. *Round Robin and Randomized Algorithms*

In the round robin [5] processes are divided evenly between all processors. Each new process is assigned to new processor in round robin order. The process allocation order is maintained on each processor locally independent of allocations from remote processors. With equal workload round robin algorithm is expected to work well. Round Robin and Randomized schemes [6] work well with number of processes larger than number of processors.

Advantage of Round Robin algorithm is that it does not require inter-process communication. Round Robin and Randomized algorithm both can attain the best performance among all load balancing algorithms for particular special purpose applications. In general Round Robin and Randomized are not expected to achieve good performance in general case.

B. *Central Manager Algorithm*

In this algorithm [10], A central processor selects the host for new process. The minimally loaded processor depending on the overall load is selected when process is created. Load manager selects hosts for new processes so that the processor load confirms to same level as much as possible. From the on hand information on the system load state central load manager makes the load balancing judgment. This information is updated by remote processors, which send a message each time the load on them changes. This information can depend on waiting of parent's process of completion of its children's process, end of parallel execution

The load manager makes load balancing decisions based on the system load information, allowing the best decision when of the process created. High degree of inter-process communication could make the bottleneck state. This algorithm is expected to perform better than the parallel applications, especially when dynamic activities are created by different hosts.

C. *Threshold Algorithm*

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

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

Initially, all the processors are considered to be under loaded. When the load state of a processor exceeds a load level limit, then it sends messages regarding the new load state to all remote processors, regularly updating them as to the actual load state of the entire system.

If the local state is not overloaded then the process is allocated locally. Otherwise, a remote under loaded processor is selected, and if no such host exists, the process is also allocated locally. Thresholds algorithm have low inter process communication and a large number of local process allocations. The later decreases the overhead of remote process allocations and the overhead of remote memory accesses, which leads to improvement in performance. A disadvantage of the algorithm is that all processes are allocated locally when all remote processors are overloaded. A load on one overloaded processor can be much higher than on other overloaded processors, causing significant disturbance in load balancing, and increasing the execution time of an application.

IV. DYNAMIC LOAD BALANCING

It differs from static algorithms in that the work load is distributed among the processors at runtime. The master assigns new processes to the slaves based on the new information collected [2] [7]. Unlike static algorithms, dynamic algorithms allocate processes dynamically when one of the processors becomes under loaded. Instead, they are buffered in the queue on the main host and allocated dynamically upon requests from remote hosts.

A. *Central Queue Algorithm*

Central Queue Algorithm [12] works on the principle of dynamic distribution. It stores new activities and unfulfilled requests as a cyclic FIFO queue on the main host. Each new activity arriving at the queue manager is inserted into the queue. Then, whenever a request for an activity is received by the queue manager, it removes the first activity from the queue and sends it to the requester. If there are no ready activities in the queue, the request is buffered, until a new activity is available. If a new activity arrives at the queue manager while there are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it.

When a processor load falls under the threshold, the local load manager sends a request for a new activity to the central load manager. The central load manager answers the request immediately if a ready activity is found in the *process-request queue*, or queues the request until a new activity arrives.

B. *Local Queue Algorithm*

Main feature of this algorithm [12] is dynamic process migration support. The basic idea of the local queue algorithm is static allocation of all new processes with process migration initiated by a host when its load falls under threshold limit, is a user-defined parameter of the algorithm. The parameter defines the minimal number of ready processes the load manager attempts to provide on each processor.

Initially, new processes created on the *main* host are allocated on all under loaded hosts. The number of parallel activities created by the first parallel construct on the main host is usually sufficient for allocation on all remote hosts. From then on, all the processes created on the main host and all other hosts are allocated locally.

When the host gets under loaded, the local load manager attempts to get several processes from remote hosts. It randomly sends requests with the number of local ready processes to remote load managers. When a load manager receives such a request, it compares the local number of ready processes with the received number. If the former is greater than the latter, then some of the running processes are transferred to the requester and an affirmative confirmation with the number of processes transferred is returned.

V. Parameters

The performance of various load balancing algorithms is measured by the following parameters.

A. Overload Rejection

If Load Balancing is not possible additional overload rejection measures are needed. When the overload situation ends then first the overload rejection measures are stopped. After a short guard period Load Balancing is also closed down.

B. Fault Tolerant

This parameter gives that algorithm is able to tolerate tortuous faults or not. It enables an algorithm to continue operating properly in the event of some failure. If the performance of algorithm decreases, the decrease is proportional to the seriousness of the failure, even a small failure can cause total failure in load balancing.

C. Forecasting Accuracy

Forecasting is the degree of conformity of calculated results to its actual value that will be generated after execution. The static algorithms provide more accuracy than of dynamic algorithms as in former most assumptions are made during compile time and in later this is done during execution.

D. Stability

Stability can be characterized in terms of the delays in the transfer of information between processors and the gains in the load balancing algorithm by obtaining faster performance by a specified amount of time.

E. Centralized or Decentralized

Centralized schemes store global information at a designated node. All sender or receiver nodes access the designated node to calculate the amount of load-transfers and also to check that tasks are to be sent to or received from. In a distributed load balancing, every node executes balancing separately. The idle nodes can obtain load during runtime from a shared global queue of processes.

F. Nature of Load Balancing Algorithms

Static load balancing assigns load to nodes probabilistically or deterministically without consideration of runtime events. It is generally impossible to make predictions of arrival times of loads and processing times required for future loads. On the other hand, in a dynamic load balancing the load distribution is made during run-time based on current processing rates and network condition. A DLB policy can use either local or global information.

G. Cooperative

This parameter gives that whether processors share information between them in making the process allocation decision other are not during execution. What this parameter defines is the extent of independence that each processor has in concluding that how should it can use its own resources. In the cooperative situation all processors have the accountability to carry out its own portion of the scheduling task, but all processors work together to achieve a goal of better efficiency. In the non-cooperative individual processors act as independent entities and arrive at decisions about the use of their resources without any effect of their decision on the rest of the system.

H. Process Migration

Process migration parameter provides when does a system decide to export a process? It decides whether to create it locally or create it on a remote processing element. The algorithm is capable to decide that it should make changes of load distribution during execution of process or not.

I. Resource Utilization

Resource utilization include automatic load balancing A distributed system may have unexpected number of processes that demand more processing power. If the algorithm is capable to utilize resources, they can be moved to under loaded processors more efficiently.

VI. Comparison

The comparison of various load balancing algorithms on behalf of the different parameters is shown in Table I.

TABLE I
PARAMETRIC COMPARISON OF LOAD BALANCING ALGORITHMS

Parameters	Round Robin	Random	Local Queue	Central Queue	Central Manager	Threshold
Overload Rejection	No	No	Yes	Yes	No	No
Fault Tolerant	No	No	Yes	Yes	Yes	No
Forecasting Accuracy	More	More	Less	Less	More	More
Stability	Large	Large	Small	Small	Large	Large
Centralized/Decentralized	D	D	D	C	C	D
Dynamic/Static	S	S	Dy	Dy	S	S
Cooperative	No	No	Yes	Yes	Yes	Yes
Process Migration	No	No	Yes	No	No	No
Resource Utilization	Less	Less	More	Less	Less	Less

VII. CONCLUSION

Load balancing algorithms work on the principle that in which situation workload is assigned, during compile time or at runtime. The above comparison shows that static load balancing algorithms are more stable in compare to dynamic and it is also ease to predict the behavior of static, but at a same time dynamic distributed algorithms are always considered better than static algorithms.

REFERENCES

[1] G. R. Andrews, D. P. Dobkin, and P. J. Downey, "Distributed allocation with pools of servers," in ACM SIGACT-SIGOPS Symp. Principles of Distributed Computing, Aug. 1982, pp. 73-83.
[2] S. Malik, “Dynamic Load Balancing in a Network of Workstation”, 95.515 Research Report, 19 November, 2000.
[3] Derek L. Eager, Edward D. Lazowska , John Zahorjan, “Adaptive load sharing in homogeneous distributed systems”, IEEE Transactions on Software Engineering, v.12 n.5, p.662-675, May 1986.
[4] H.S. Stone, “Critical Load Factors in Two-Processor Distributed Systems,” IEEE Trans. Software Eng., vol. 4, no. 3, May 1978.
[5] Zhong Xu, Rong Huang, "Performance Study of Load Balancing Algorithms in Distributed Web Server Systems", CS213 Parallel and Distributed Processing Project Report.
[6] R. Motwani and P. Raghavan, “Randomized algorithms”, ACM Computing Surveys (CSUR), 28(1):33-37, 1996
[7] Y.Wang and R. Morris, "Load balancing in distributed systems," IEEE Trans. Computing. C-34, no. 3, pp. 204-217, Mar. 1985.
[8] M. Zaki, W. Li, and S. Parthasarathy. “Customized dynamic load balancing for a network of workstations”. Journal of Parallel and Distributed Computing: Special Issue on Performance Evaluation, Scheduling, and Fault Tolerance, June 1997.
[9] S.P. Dandamudi, “Sensitivity evaluation of dynamic load sharing in distributed systems”, IEEE Concurrency 6 (3) (1998) 62-72.
[10] P. L. McEntire, J. G. O'Reilly, and R. E. Larson, Distributed Computing: Concepts and Implementations. New York: IEEE Press, 1984.
[11] L. Rudolph, M. Slivkin-Allalouf, E. Upfal. A Simple Load Balancing Scheme for Task Allocation in Parallel Machines. In Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures, pp. 237-245, July 1991.
[12] William Leinberger, George Karypis, Vipin Kumar, "Load Balancing Across Near-Homogeneous Multi-Resource Servers", 0-7695-0556-2/00, 2000 IEEE.