International Journal of Robotics Applications
Volume 1, Issue 1, 2011, pp-01-02
Available online at: http://www.bioinfo.in/contents.php?id=110

Load Balancing in Amoeba

[1]S.R. Durugkar and [2]Pritam Jape
[1]*SCOE, Pune*
[2]*KBP, Poly, Kopargaon*
e-mail: santoshdurugkar@gmail.com, pritamaditya_77@rediffmail.com

Abstract—In this paper we represent the use of load balancing using job initiation and process migration came out on Amoeba. And also indicate the need for a load balancing facility in a distributed system to improve system performance, e.g., the average response time of processes. A number of load balancing algorithms, including the bidding and neighboring algorithms, have been studied in this work. A comparison between these algorithms under various conditions is presented, which indicates that in a system with few computers a centralized algorithm outperforms a distributed one and job initiation plays an important role in a load balancing scheme.

Keywords: Amoeba, Load Balancing, Random, distributed.

I. INTRODUCTION

A distributed computer system with 10-100 of computers connected by high-speed networks has many advantages over a system that has the same computers.

One of the important advantages of a distributed system is its provision for efficient resource sharing. This indicates that under the same conditions a distributed system should provide much better service than a traditional system in terms of performance and reliability. In order to achieve the performance potential of a distributed system, a dynamic load balancing facility is required. This facility monitors load variation to detect load imbalance, and then takes action to balance the load. Placing, replacing (redistributing) and replicating some objects in a system are the possible actions which can be adopted by a load balancing facility.

In this paper, we presenting the few points of load balancing, which only takes Processes into account. A considerable number of research projects on load balancing has been carried out over many years due to the potential performance gain from this service.

Most of the research has focused on load balancing algorithms and load measurement. The aim is to find general effective load balancing algorithms with overhead as low as possible. The methods used in the past range from analytic study plus simulation to implementation plus testing. Due to the difficulty of implementing a load balancing facility in a distributed system, the majority of studies fall into simulation.

II. LOAD BALANCING FACILITIES IN AMOEBA

Amoeba wasn’t designed and implemented with dynamic load balancing in mind because the designers advocate the processor-pool model which assumes the number of processors is more than the number of processes. However, current computing environments typically contain autonomous workstations connected through networks, which requires load balancing for its performance.

A. The Structure of Load Balancing Facility in Amoeba

In order to support both preemptive and nonpreemptive load balancing strategies, we studied the load balancing facility as two sorts of processes: load balancers and process migration servers.

A load balancer makes decisions on *when to move which process to where,* while, a migration server carries out the decision made by a load balancer to move processes between computers. Therefore, the load balancer plays the policy role in this facility. On the other hand, a process migration server supplies the mechanism in the load balancing facility. This separation between policy and mechanism offers a flexible structure for our experimental study of various load balancing algorithms. The process migration server provides the same service, regardless of the differences between policies.

B. Process Migration on Amoeba Operating system

An Amoeba process may have multiple threads, each of which is an object that may be scheduled by the kernel. During a process migration, all threads of the process must be suspended and moved from the source to the destination. The execution state of a process is □arshaled into messages and sent to the destination to reconstitute. Besides the execution state, the memory space of a process which contains other states must be transferred as well. Although a number of techniques are available for transferring the memory space of a process, such as lazy copying and a shared file server, at this stage we have limited ourselves to a straightforward implementation – direct copying, because Amoeba does not support virtual memory. Apart from migrating execution state and memory space of a process, the communication state of the process has to be transferred to the destination. In Amoeba, each thread of the process has its own communication state which remembers the stage of these ongoing RPCs and the role the thread plays (client or server) in those RPCs.

These communication states, including the capabilities of its RPC partners, are kept in the kernel.

III. Load Balancing Algorithms

There are a large number of load balancing algorithms. These algorithms can be classified into different categories. In the first stage, we studied some simple algorithms which form two classes: nonpreemptive and preemptive. In the non-preemptive class, we have a number of job initiation algorithms:

A. Centralized Initiation

In the centralized approach, there is one load balancer in the whole system which is responsible for assigning new jobs to computers. When a new job arrives, the load balancer relies on its periodically collected load information to determine the machine to execute the new job and starts the job on that machine.

B. Distributed Initiation

There is a load balancer on each computer, which broadcasts its load to others whenever it detects a load change in its computer. When a new job arrives, the load balancer, based on its own load, the received load information and the age of these information, determines which computer should host the new job and starts the job on that computer. This algorithm tries to start a job on a destination without negotiation. Otherwise, the job is started locally. The aim of using the filter is to reduce the number of RPCs and speed up the decision-making process.

C. Random Initiation

This is also a distributed method. When a new job arrives at a computer, the local load balancer randomly selects a computer to run the job if it is overloaded at that moment. Otherwise, the job is started locally.

D. Central

There is one load balancer running computers to obtain system load information.

When the load balancer detects load imbalance instances, it selects one of the computers with the lowest load to accept a process from an overloaded machine. In the system, which regularly polls the other.

E. Random Selection

Within this algorithm, when a computer becomes overloaded, it randomly selects another computer to shift its load, no matter whether the computer is under loaded or not.

F. Neighbor

As described in above, each computer regularly sends request messages to its neighbors, containing its current load information. When the neighbor receives this message and accepts the request, then these two computers become a pair. Later, based on the load difference between them, the computers decide to shift some load from one to another. After that, the pair is broken.

G. Broadcast

This is a server-initiated algorithm. Load balancers exchange their load information whenever the load is changed. When a computer becomes underloaded (in our case, when it becomes idle), its load balancer checks whether there are overloaded machines from the received load information. If so, it first selects a computer and a process that is running on that computer, then calls the process migration server to move this process.

IV. Conclusion

In this paper, we studied the various load balancing algorithms on the Amoeba system. After studying the various load balancing algo. we can say that a centralized load balancing strategy is more effective than a distributed one when there are 10 - 20 computers in a system. This is due to the fact that a centralized approach creates much less overhead than a distributed one. Because the authority to make load balancing decision is concentrated in one place, it avoids possible conflicts between different decision makers, and the negotiation overhead which is used by most distributed algorithms to avoid conflicts.

References

[1] A. Barak, A. Shiloh, and R. Wheeler. Flood Prevention in the MOSIX Load-Balancing Scheme. *TCOS Newsletter,* 3(1), 1989.
[2] R.M. Bryant and R.A. Finkel. A Stable Distributed Scheduling Algorithm. *In Proceedings* of *the* 2nd *International Conference on Distributed Computing Systems,* April 1981.
[3] T.L. Casavant and J.G. Kuhl. A Taxonomy of Scheduling in General-purpose Distributed Computing Systems. *IEEE Tkans. on Software Eng.,* 14(2), Feb. 1988.
[4] F. Douglis and J. Ousterhout. Transparent Process Migration: Design Alternatives and Sprite Implementation. *Software and Ezperience,* 1991.
[5] P.K. Sinha, Distributed operating System.