Mechanism for Implementation of Load Balancing using Process Migration

Narayan A. Joshi
Assistant Professor
Computer Science Department
Institute of Science & Technology for Advanced Studies & Research (ISTAR), Vallabh Vidyanagar,

D.B. Choksi
Professor & Director
Department of Computer Science & Applications,
Sardar Patel Univerisity, Vallabh Vidyanagar,
Gujarat, India

ABSTRACT
The feature of load sharing or load balancing involves migration of running processes from highly loaded workstations of a network to the lightly-loaded or idle workstations of the network. This paper describes load balancing techniques to share the workload of the workstations belonging to a particular network to gain better performance from the overall network. The mechanisms of load information collection, determining the idle workstations as well as highly-loaded workstations, and transferring the load from one workstation to another workstation are represented in this paper. The paper describes a dynamic load balancing algorithm to achieve runtime performance gained by managing the jobs as they arrive. We have introduced the agent-based approach towards load balancing in this algorithm. The benefit of load balancing can be achieved through either the server-managed load balancing or the client-initiated load balancing, both of them having specific advantages and drawbacks. In this paper, both of these load balancing approaches have been supported by the presented algorithm in order to gain the benefits of reliability, fault tolerance and efficiency and high performance.

General Terms
Load balancing, load sharing, process migration, Load estimation

Keywords
Server managed load balancing, client initiated load balancing.

1. INTRODUCTION
Use of distributed computing systems is increasing day by day because of the availability of powerful hardware at lower cost and innovations in networking and communication technologies. Some of the primary important benefits of distributed systems are the strength of resource sharing to provide the users with a wealthy collection of resources which are usually spread across the member workstations of a particular network and the load balancing of network's workstations. The total computational capacity of the networked workstations may on paper be very much larger than that of the original uniprocessor computing system. It is generally observed that, the hardware has been upgrading more quickly than the supporting system software and the operating systems. Repeatedly each workstation-user will execute their corresponding applications on their particular system, which gradually becomes either overloaded or almost entirely idle. It is frequently observed that, in a computing environment with a numbers of hosts connected by networks, there is high probability that some of the workstations are heavily loaded, while others are almost idle. This in effect causes utilization of only a fraction of the network's theoretical power.

The last decade has observed the development of tightly coupled parallel multi-processor systems which have been trendy in high performance computing environments. By connecting simpler, smaller and cheaper processors and other components, a computing system may be assembled with the same processing capacity as that of one utilizing a highly complex, large and powerful processor but for a fraction of the price. There exist numbers of drawbacks with such parallel computing systems like a limited support or lacking of fault tolerance and limited or negligible support in hardware upgrading.

On the opposite side, the advancements in microelectronic technology and communication technology has resulted in the availability of fast, inexpensive processors and cost-effective "loosely-coupled systems" known as "Distributed systems"; in which software components located at networked nodes communicate and cooperate their actions to achieve the benefits like – information sharing, resource sharing, better flexibility, higher throughput and higher reliability. It is practically possible to either insert additional workstations to or remove existing workstations from the distributed systems. It means that the limitations of the parallel computing architectures do not exist with distributed systems. [13], [8]

Thus, the so far made discussion suggests that performance gains may be achieved by transferring processes from the currently heavily loaded workstations to either the idle workstations or the lightly-loaded workstations. Such a technique of computing-power sharing for the purpose of improving the performance of an overall distributed system by redirecting the workload among the available hosts is known as 'Load Balancing'. [2], [3], [5], [8], [13]

Section 2 briefly describes the mechanism to implement the load balancing feature. Section 3 focuses on the theoretical aspects and our algorithmic approach to achieve the solution. Finally, the section 4 gives the concluding remarks.

2. MECHANISM
The problem of load balancing has been studied using a number of different approaches for long time. The earlier works primarily focused on static load balancing. In those studies, process transfer decisions are made probabilistically or deterministically without taking into consideration the current state of the system. Static algorithms take into consideration only the system's average behavior and ignore the current state of the system; static load balancing is simple and effective when the workload can be sufficiently well

characterized beforehand, but it fails to adjust to the fluctuations in the system load. In distinction, in order to gain greater performance benefits, dynamic load balancing attempts to balance the system load dynamically as jobs arrive. In this paper, we suggest the dynamic algorithmic approach to provide load balancing.

Out of the probabilistic and deterministic load-balancing approaches, we follow here the deterministic approach in order to gain better load-balancing though it is difficult to optimize and implement.

The algorithms based on periodic or non-periodic load information exchange, provide good performance, and among the periodic and non-periodic policies, the algorithms that use a distinguished agent to collect and distribute load information cut down the overhead and scale better. The performances of all hosts, even those originally with light loads, are generally improved by load balancing. In this chapter we have represented such an agent-based load information collection and distribution techniques on periodic as well as non periodic policy basis.

In this paper, we represent algorithm to provide significant reduction in the load of heavily-loaded workstations in order to provide substantial performance improvements. We have represented the two techniques: the server managed load balancing to serve periodic load-balancing policy and the client-initiated load balancing to serve the non-periodic load balancing policy.

Section 4 presents conclusion and directions for future work. [7], [12]

3. THE ALGORITHM

We provide the both types of feature-algorithms: centralized and distributed. In order to provide the centralized dynamic load-balancing, we have established a central server that is responsible not only to manage the workload information about all the workstations of a particular distributed system, but also it dynamically executes the load-balancing policy, determines the overloaded and under loaded workstations, and instructs the overloaded workstations to carry out process migration to reduce its load. The centralized approach of workload management makes efficient load-balancing decisions as the server itself holds every client-workstations' load information. [4], [12]

The major weakness associated with the centralized load balancing approach is that of performance bottleneck of the central load management server and reliability. Overloading on the centralized load-balancing server may cause either weak load balancing or delay in the decision making of load transfer from one workstation to the another one. On the other side, in case of the failure of the centralized server, there will occur an absence of decision making of the load-balancing. Therefore, here we present a distributed module for load balancing, which is responsible to periodically observe the local workstations load and decides whether to perform load balancing or not, this approach works on the co-operative basis.[4], [12]

The major steps of our algorithm are suggested in brief here:

1. Create a software module 'LoadPortal' with the following functions: (a) It periodically calculates the particular client's load according to the prespecified load-calculation policy; and (b) sends the above calculated workstation's load to the load management server. Install this LoadPortal software module on each client in the network. [1]

2. On the server side, create a software module 'LoadServer' with the following functionalities: (a) it iteratively listens for the load-details which are sent by all the workstations of the network in a concurrent way. (b) According to the specified load-balancing policy, the module concurrently determines: (i) the client having the higher load-volume which is more than the specified upper load-threshold; the server-module will make efforts to reduce the workload of this highly loaded client say it client_A. (ii) the idle workstation or the workstation having least workload which is less than the specified bottom load-threshold; the server module will make efforts to migrate specific process(es) from the highly loaded workstation to this lightly loaded workstation say it client_B.

3. On the server side, create a software module 'LoadBalancer' with the following functionalities: (i) It runs iteratively and works in a concurrent manner. (ii) It sends an advisory order or an instruction 'Migrate specific process(es) to the lightly loaded workstation client_B' to the highly loaded workstation client_A. (The client_A and the client_B have been determined in steps 2-b-(i) and 2-b-(ii) respectively.) Thus, the current step implements the approach of server-managed load balancing.

4. On the opposite side, on the network's workstation, implement and execute one more software module having the following capabilities: (i) it iteratively listens for the advisory orders or instructions which are sent by the 'LoadBalancer' (step 3). (ii) it determines particular process(es) for their migration to remote workstation, say such process(es) as 'victim process(es)'. (iii) it performs migration of the these selected victim processes using the 'ProcessMigrator' module (see step-6) from the workstation client_A to the workstation client_B; thereby reducing the load of client_A.

5. In addition, the software module 'LoadPortal' (step 1) on the network's workstations, also runs one more thread of execution in order to perform client-initiated load balancing; this thread periodically observes the client's load and may decide to reduce its load because of the reasons like spontaneous or a sudden increase of the workstation's load above the upper load-threshold; or the workstation itself may wish to migrate its some of the specific process(es) to remote workstations to gain the benefits like fault-tolerance and process control. For such reasons, this thread may decide to obtain the details of the idle workstation or the least-loaded workstation say client_B with the help of the step 2-b-ii; and migrates the particular process(es) to the client_B using the 'ProcessMigrator' module (see step-6). Thus, the current step implements the approach of client-initiated load balancing.

6. Create and implement one more module 'ProcessMigrator' with the following features: (i) it iteratively listens for the requests to perform process migration; the request mainly comprises of the information like what is to be migrated i.e. the process's id, and where the migration has to take place, i.e. the information about the destination workstation. (ii) after receiving the migration order, the ProcessMigrator module performs migration of the desired process to the

destination workstation by using the mechanisms 'process checkpointing and restore'. Process checkpointing involves capturing the current state of the desired process, migration of such captured state to the remote workstation, restoration of this particular process state on the remote workstation and resumption of the process on remote workstation. The desired process's state capturing mechanism involves gathering information like the memory regions occupied by the process, the details of system call which is being executed by the process, the process credentials possessed by the process and many other details.

[2], [5]-[7]

4. CONCLUSIONS

An attempt is made here to design algorithms for both types of approaches: the client-initiated load-balancing (distributed approach) and the server-managed load-balancing (centralized approach). The algorithms follow the deterministic aspect to achieve load balancing. We wish the solution suggested here will be helpful to the process migration systems.

5. REFERENCES

[1] Bovet, D. P. and Cesati, M.: "Understanding the Linux Kernel", 3rd editon, O'Reilly publication, 2005; pp-44-46, pp-90-95

[2] Corbet, J., Rubini, A. and Kroah-Hartman, G.: "LINUX Device Drivers", 3rd edition, O'Reilly publication; pp-412-415, pp-419-422

[3] Douglis, F. and Ousterhout, J.: "Process migration in the sprite operating system"; Proceedings of the 7th international conference on distributed computing systems, IEEE, Berlin, West Germany, September 1987, pp. 1825

[4] Gupta, V., Harchol-Balter, M., Scheller-Wolf, A. and Yechiali, U.. "Fundamental Characteristics of Queues with Fluctuating Load." Proceedings of ACM SIGMETRICS 2006 Conference on Measurement and Modeling of Computer Systems . Saint Malo, France, June 2006, pp. 203-215

[5] Joshi, N. A. and Choksi, D. B.; "Process Forensic For System-call Details on Linux Platform"; International Journal Of Computer Applications in Engineering, Technology & Sciences; November-2009; pp-510-512

[6] Joshi, N. A. and Choksi, D. B.; "Checkpointing Process Virtual Memory Area on Linux Platform"; International journal of Emerging Technologies and Applications in Engineering Technology and Sciences; June-2010; pp-42-44

[7] Joshi, N. A. and Choksi, D. B.; "Checkpointing Process VMA for process migration"; Journal of Emerging Trends in Computing and Information Sciences; Volume 2 Special Issue February-2011; pp-7-10

[8] Kozuch, M. and Satyanarayanan, M.: "Internet suspend/resume"; Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, IEEE CS Press, 2002, pp. 40–46

[9] Milojicic, D., Douglis, F., Paindaveine, Y., Wheeler, R. and Zhou, S.: "Process Migration"; ACM Computing Surveys; 32(3):241-299, 2000

[10] Nichols, D.: "Using idle workstation in a shared computing environment"; Proceedings of the eleventh ACM symposium on operating system principles; ACM; November – 1988; pp 512

[11] Schroeder, B. and Harchol-Balter, M.; "Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness." Invited to: Cluster Computing: The Journal of Networks, Software Tools, and Applications , volume 7, Issue 2, April 2004, pp. 151-161

[12] Sinha, P. K.; "Distributed Operating Systems – Concepts and descign"; PHI publications

[13] Smith, J. D.; "Fault Tolerance using Whole-Process Migration and speculative Execution"; 2003 - M. S. Degree Thesis – California Institute of Technology.