1998 International Computer Symposium
Workshop on Software Engineering and Database Systems
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

The Study of Centralized Load Balance on Distributed Shared Memory Systems

Yi-Chang Zhuang, Ce-Kuen Shieh and Tyng-Yen Liang

Department of Electrical Engineering,

National Cheng Kung University, Tainan, Taiwan, R.O.C.

Email: {zhuang,shieh,lty}@ee.ncku.edu.tw

Abstract

Distributed Shared Memory facilities programmers with the similar programming interface of shared memory. However, it suffers the problem of load imbalance especially when the composite computers are not equipped with the same class of processor or when the workload of program is not partitioned equally among tasks. As a result, it is urgent to achieve load balance on a DSM system. In this thesis, we design a centralized dynamic load balance mechanism for an available software DSM system and the experiments show that it improves the system performance significantly over that with no load balance involved.

1. Introduction

Distributed Shared Memory (DSM) [6][7][8] provides a virtual shared space under the distributed environment via the software or hardware simulation. With the virtual shared memory, programmers can develop parallel programs in the shared memory programming model on a multiple computer network, which is the same as they do on a shared memory multiprocessor system. Besides, the Distributed Shared Memory system reduces the complexity of parallel processing under a distributed computing environment. However, there are some important issues that are critical to the performance of a DSM system. Among all, load balance [1][3][4][9][10][11][12] is the most important.

Generally, there are two basic strategies to reach the goal of load balance. One is static scheduling, and the other is the dynamic load balance. The philosophy of the static scheduling is to design an algorithm that can find the optimal or nearly optimal solution to distribute the workload in advance according to the status of the system. As a result, programmers can make best use of system computing power and shorten the execution time of parallel applications. In some specific cases, such as that the system resource configuration is always the same when one parallel program runs, it really performs as we expected. However in all the other cases, its performance is not as good as that in previous cases, especially in the situation that the runtime information of the programs cannot be described in specific pattern. On the contrary, dynamic load balance delays the actions of the runtime system information collection and analysis and puts them into action concurrently with the execution of the applications. Consequently, it can take action to redistribute the workload according to the state of the system at that time and it can detect and deal with the load imbalance resulted from the fluctuation of the program on the instant.

In our research, we not only focus on dynamically balancing the workload but we also pay attention to minimize the communication overhead incurred by the workload redistribution. Because software DSM systems consists of a cluster of computers connected by network, the data which is not located on the node must be fetched from the remote nodes through the underlying network and the consistency of memory pages is maintained by message passing. It is apparent that if we only focus on balancing the workload without taking the data locality into consideration, many network messages will be presented on the network after the workload redistribution. To solve this problem, we had proposed a Dependency-Driven Load Balance (DDLB) algorithm designed for the iterative-barrier type of parallel applications and it was implemented on a distributed load balance

mechanism on a software DSM system developed in our laboratory – Cohesion [6][7]. From the proposed paper [1], it shows that the DDLB algorithm is not only able to keep the load of a DSM system well-balanced but it also can minimize the communication overhead resulted from the load balance. However, because distributed load balance takes only local system information into consideration in most cases and the work to balance the load is carried out on every computer independently without any negotiation and compromise, it seldom performs the best. In worst cases, it may waste the system resource on load balance and degrade the system performance. In addition, the design logic and implementation of distributed load balance is more complicated and subject to errors than that of centralized load balance. On the other hand, in the centralized load balance all the system-related information is gathered on one computer. This computer makes use of the information to search the optimal or nearly optimal workload distribution and acts as the manager to direct all the other computers to balance the workload. Consequently, centralized load balance is likely to outperform the distributed one in most cases except that centralized load balance will more likely to be the bottleneck of the whole system as the scale of the system is getting larger and larger. But, the proposed software DSM systems are seldom in large scale and thanks to the rapid hardware advance, the computing capability of the computer is more powerful than before. So, we implement the DDLB algorithm on a centralized load balance.

The remainder of this paper is organized as follow. In next chapter, we discuss the design issues and consideration of a load balance mechanism on a DSM system. Then we illustrate the detail of the implementation of our centralized dynamic load balance in chapter 3. In chapter 4, we show and analyze the experimental results of the performance evaluation. Finally, the conclusions and suggestions for the centralized dynamic load balance on a DSM system are given in the last chapter.

2. Design Issues

Basically, a dynamic load balance algorithm consists of four elemental components: a transfer policy, a location policy, a selection policy and an information policy [4][10][11]. Similarly, all of these four policies are critical to the efficiency of our centralized dynamic load balance mechanism and take effect on the performance improvement of the DSM system.

2.1Transfer Policy

Typically, the transfer policy determines whether a node is suitable for participating in the load redistribution at the execution time of the parallel programs. It is usually based on the load function, such as ready queue length, predefined in the design of the algorithm. Researches [4][12] have shown that even simple load function is effective in representing the load indices of the system. However, such a simple policy as the one described above usually performs well under the situations that the computing power of all constituent processors are almost the same or the workload of the parallel programs is equally distributed between every task. But, as time revolves, it is common that a DSM system is composed of the processors with the same hardware architecture but with different computing power. Consequently, a load function may be too simple to be adopted in representing the load indices of the system. What is worse, when the workload is not equally partitioned among the tasks, the dynamic load balance mechanism may make wrong decisions and degrade the system performance. As a result, in order to evaluate the workload correctly, we get rid of all simple policies and design a more complicated load function which takes not only the workload of every tasks but also the computer power of processors into consideration. Besides, the transfer policy of our centralized load balance mechanism is designed to determine whether a node is overloaded to be the sender of the task redistribution and whether redistributing workload will improve system computing resource utilization and shorten the execution time.

2.2Location Policy

Secondly, the location policy tries to find a node, which is capable of consuming the workload transferred from the sender node and will not extend the execution time after task transfer. In

general, the location policy can be categorized based on the complexity of algorithm it adopted and the implementation of dynamic load balance, such as random transfer, polling. In consideration of the overhead introduced to the system and the design of our dynamic load balance mechanism, we adopted the centralized coordinator to locate a suitable node for load sharing. With enough system information available, the centralized coordinator can correctly locate a suitable node as the destination. And our centralized load balance mechanism can transfer the task selected by selection policy on the sender node to the destination node.

2.3Selection Policy

As regards the selection policy, it is responsible for selecting a suitable task to transfer. Once the transfer policy determines a node to be a task sender and the location policy finds a competent destination node, the centralized dynamic load balance mechanism starts to perform the selection policy. Generally speaking, the criterion of task selection is to select a task that is not related to the sender node or less related to the sender node and minimize the overhead incurred by the task transfer. On a DSM system, the relationship between a task and a node is evaluated according to the information that how many memory pages accessed by the task are located on the node. That is, the more memory pages accessed by a task a node owns, the more close relationship between the task and the node, because a software DSM system is constructed by grouping a cluster of computers connected by the underlying network. As a result, if the load balance mechanism happens to migrate out a task that is more related to the node than the others are, it will certainly result in more communication overhead than the others do after the task migration. And it is probable that the communication message may be too much to degrade the system performance. In view of this phenomenon, our centralized load balance mechanism adopt the proposed Dependency-Driven Load Balance (DDLB) algorithm which was proved to be able to reduce the communication overhead incurred by dynamic load balance mechanism. In the design of the DDLB algorithm, it takes both the relationships between the source node and the transferred task and between the destination node and the transferred task into account. Therefore, our centralized dynamic load balance mechanism will be able to improve the system performance without incurring too much communication overhead.

2.4Information Policy

Finally, the information policy decides what information should be collected and when the information of the system is to be collected. The information used as the parameters of the dynamic load balance mechanism can be the ready queue length, the history of system workload, the mean response time, etc. However, collecting too many kinds of information may not significantly improve the effect of dynamic load balance. To make the matters worse, it probably turns out to be another burden on the DSM system. According to the requirements of the three policies described above, the computing power of every processor, memory pages accessed by every task and execution time of every task are all indispensable to our centralized dynamic load balance. Besides, because our centralized dynamic load balance is designed to solve the load imbalance problem of iterative-barrier type of applications, we prefer to collect these kinds of information and report them to the centralized coordinator at the end of each iteration to eliminate the load imbalance phenomenon in no time.

3. Implementation

3.1Cohesion

We have discussed the considerations and the design issues of our centralized dynamic load balance mechanism in last paragraph. The following is the detailed implementation. Since it is designed and implemented on our available software DSM system, Cohesion, which was designed and implemented in our laboratory. We will first give an introduction on Cohesion.

Cohesion is a multithreaded software DSM system built on top of Intel 80x86 personal computers. It facilitates programmers with a convenient parallel programming environment with the user-level object-oriented thread system

which resembles the programming interface of PRESTO [2], a user-level thread package that was originally designed and implemented by University of Washington. Besides, there are also a communication subsystem and a DSM layer in Cohesion. The communication subsystem provides a reliable and efficient data communication service between computers.

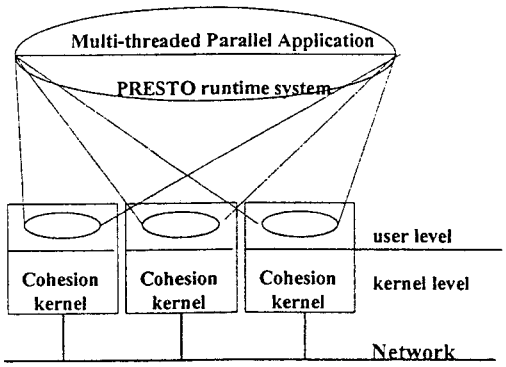


Fig.1 System architecture of Cohesion

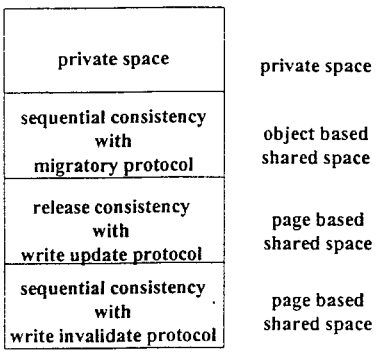


Fig. 2 Memory spaces in Cohesion

The DSM layer acts as the memory coherence manager and maintains the consistency of the virtual shared memory space in four kinds of protocols. First, the private space is provided to record local data of each computer and is not shared with the other computers. As a result, there is no network communication message resulted from the modifications in this address space. The object based shared space is implemented sequentially consistent with migratory protocol. Consequently, the data in this space, such as the synchronization object, will get free of false sharing. For the page based shared space, it can be further divided into sequential consistent memory and release consistent memory. While the sequential consistent memory is maintained with write invalidate protocol, the release consistent memory is maintained with write update protocol [5][6]. Since Cohesion provides an abstraction of a global shared address space, writing parallel programs on it is like writing parallel programs on shared memory multiprocessors.

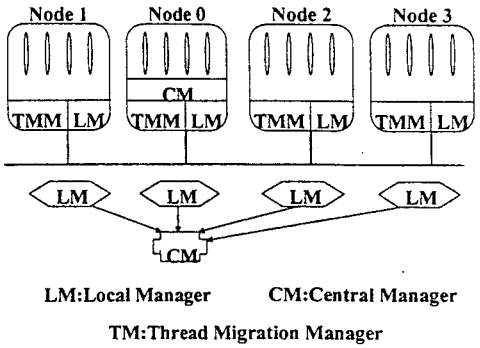


Fig. 3 Components in our mechanism

3.2Centralized Dynamic Load Balance

We have discussed the design issues of the centralized dynamic load balance on Cohesion. Now it is the time to explore the implementation of the centralized dynamic load balance to understand how it can solve the load imbalance problem on a DSM system.

As we have mentioned earlier, the central part of the dynamic load balance is composed of four important policies and detail implementation is shown in last figure.

The Local Manger (LM) and Thread Migration Manager (TMM) are implemented in the load balance mechanism on each computer, while the Central Manager (CM) is just implemented only on one computer, which is usually located on the root node. The LM module is responsible for collecting the various kinds of local system information and reporting them to the CM in the course of execution. Besides, it also adjusts the copyset after the task redistribution to avoid false sharing arises. After receiving the information sent from all other nodes, the CM module is activated to redistribute the workload of the system if necessary. At last, the TMM module migrate the tasks from the source node to the destination node according to the calculated results of the CM module. However,

the tasks will not be migrated until they are finished from the point of the synchronization in our recent implementation. This kind of implementation is to simplifies the design of the dynamic load balance because there is no additional overhead, such as message forwarding, necessary to have the entire system operational. The following is the flow chart of the CM module.

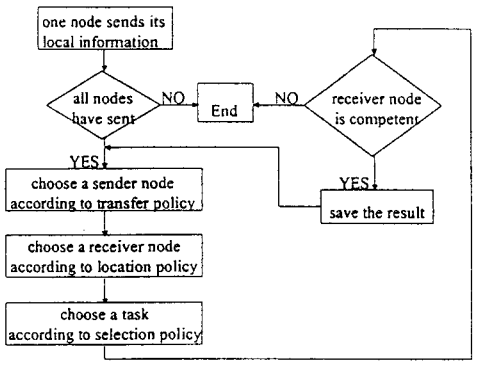


Fig. 4 Flowchart of the CM module

3.3Data Dependency Load Balance(DDLB)

Another key point of our centralized load balance is the DDLB algorithm that minimizes the communication overhead resulted from task migration. Let's take an example in next figure. Suppose that there are two nodes on the system and three tasks executed on each node. According to the load balance algorithm, we determine that the sender node is node 1 and the receiver node is node 0. Then we make use of the data access pattern of every task to calculate the intra-dependency and inter-dependency of every task. In the DDLB algorithm, intra-dependency is defined as the amount of pages accessed by a task and its sibling tasks on the same node. Inter-dependency is defined as the amount of pages accessed by a task on source node and the task on destination node.
According to the definition, it is clear that inter-dependency is the source of communication message before task migration while intra-dependency is the source of communication overhead after task migration. As a result, the dynamic load balance will select thread 4 on node 1 for task migration.

node 0		node 1	
	access pages		access pages
thread 1	1 8	thread 4	1 2 3 6 8 9
thread 2	3 7 8 9	thread 5	1 2 5 7
thread 3	1 5 6 8	thread 6	1 2 4 5 7

Thread	intra-dependency	inter-dependency
4	1 2	1 3 6 8 9
5	1 2 5 7	1 5 7
6	1 2 5 7	1 5 7

Fig. 5 An example of DDLB

4. Experimental Results

In this section we present the experimental results of the centralized dynamic load balance on DSM system.

4.1Experimental Environment

The test bed is a multithreaded software DSM system based on Intel 80x86 series of personal computer, 486DX-33 and Pentium-90, and the underlying network is 10Mbps Ethernet.

4.2Experimental Results

We evaluate the performance improvement of our dynamic load balance in two applications, Successive Over-Relation (SOR) and N-Body. SOR is a linear equation problem found in the field of engineering phenomena. It uses the algorithm of the iterative method to approximate the solution of a partial differential equation. The idea of the SOR algorithm is to compute a better approximation to the true solution in accordance with the solution in previous iteration. The function adopted in our testing program is the average of its neighbors (above, below, left and right). N-Body is a force calculation problem in the field of astrophysics. It calculates the interacting force among particles in the rule of the Newton theorem. The following shows the execution time of our testing programs and its speedup with or without our centralized dynamic load balance mechanism applied.

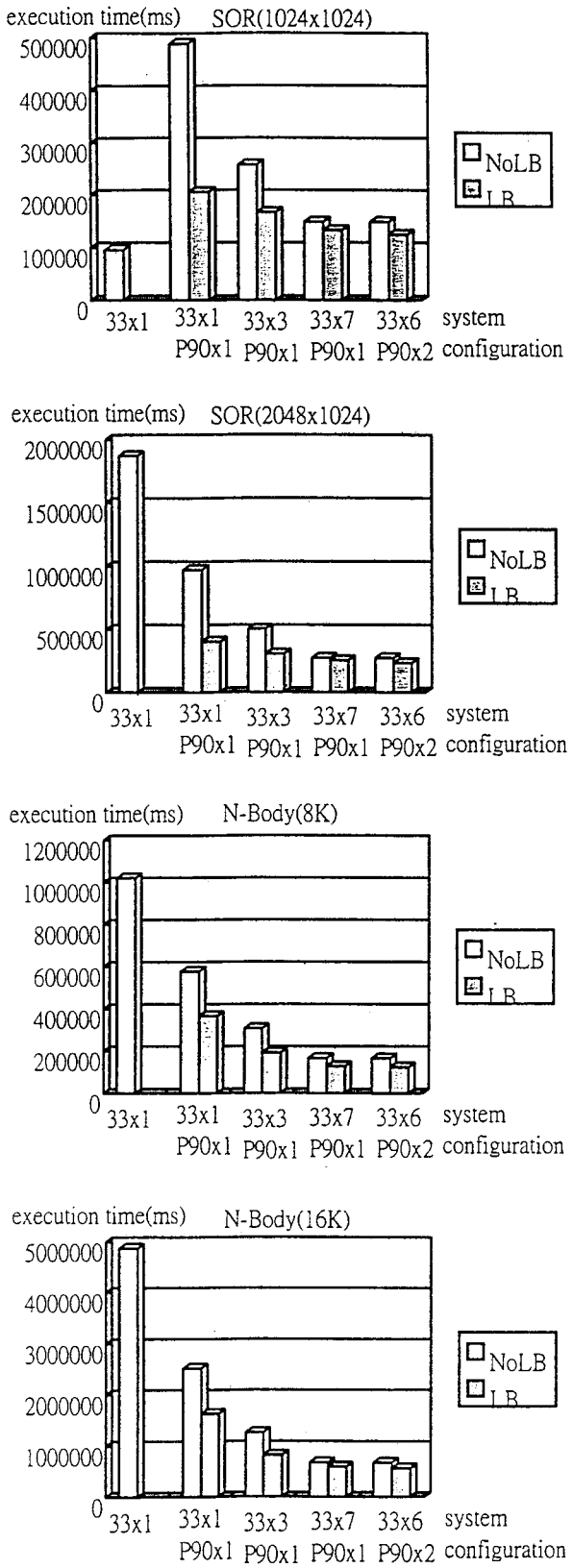
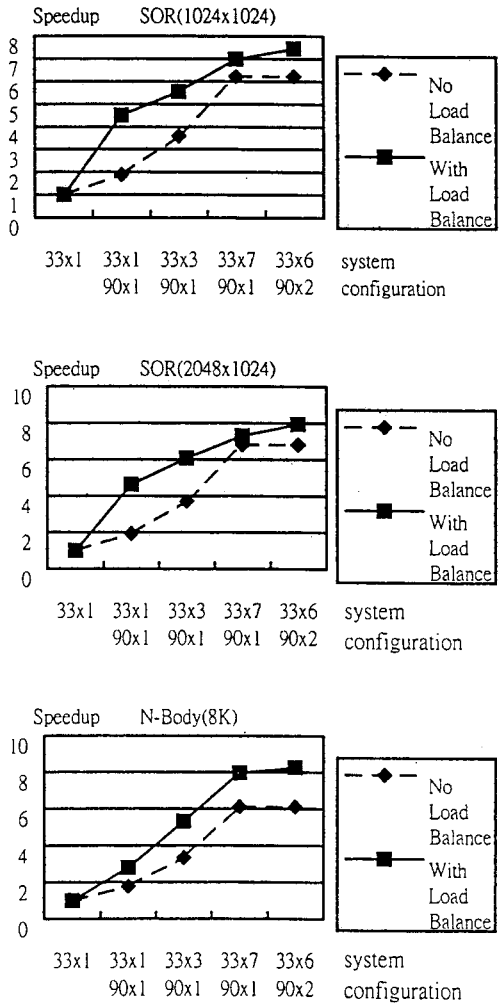


Fig. 6 Execution time of SOR and N-Body

When writing programs, we are not aware of the system configuration at all times. As a rule, we partition the workload of the program into every task equally. However, as shown in our experimental results, it results in load imbalance phenomenon in our experimental programs since the computing power of the Pentium series processor is more powerful than that of the 486 processor. The computing power of Pentium processor is wasted waiting for the 486 processor without our centralized dynamic load balance. But when the dynamic load balance mechanism is integrated into the multithreaded software DSM system, the improvement on performance with our centralized dynamic load balance is better than that without load balance. Next figure show the speedup of our testing programs with or without our centralized dynamic load balance mechanism.



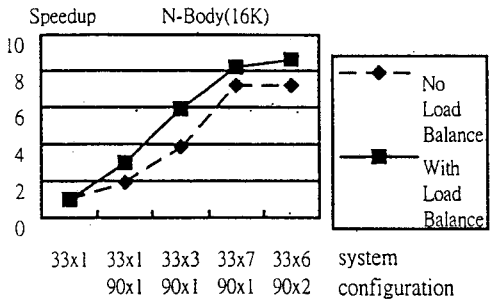


Fig. 7 Speedup of SOR and N-Body

5. Conclusions

In this thesis, we have presented a centralized dynamic load balance mechanism for a multithreaded software DSM system. The purpose of our dynamic load balance is to achieve load balancing efficiently and transparently. Since communication plays an important role in the system performance, our dynamic load balance also aims at minimizing the communication overhead incurred after the task transferring. According to the experimental results, it is clear that there is a significant system performance improvement over no load balancing effort. With the help of the centralized dynamic load balance, we can make full use of the computing power of all processors available when running a parallel program without introducing too much overhead.

6. References

[1] An-Chow Lai, Ce-Kuen Shieh, Jyh-Chang Ueng, Yih-Tzye Kok, and Ling-Yang Kung. "Load balance in Software Distributed Shared Memory Systems". IEEE International Performance, Computing, and Communications Conference, Arizona, U.S.A. Feb. 5-7, 1997.

[2] Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy. "PRESTO: A System for Object-Oriented Parallel Programming." Software Practice and Experience, vol. 18, no. 8, pp. 713-732, Aug. 1988.

[3] Derek L. Eager, Edward D. Laxowska and John Zahorjan. "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing." ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Aug. 1985.

[4] D. L. Eager, E. D. Lazowska and J. Zahorjan. "Adaptive Load Sharing in Homogeneous Distributed Systems." IEEE Transaction on Software Engineering, 1986.

[5] John B. Carter. "Efficient Distributed Shared Memory Based on Multi-protocol Release Consistency." Ph.D. dissertation, Rice University, Sept. 1993.

[6] Ce-kuen Shieh, An-Chow Lai, Jyh-Chang Ueng, Tyng-Yeu Liang, Tzu-Chiang Chang, and Su-Cheong Mac. "Cohesion: An Efficient Distributed Shared Memory System Supporting Multiple Memory Consistency Models." AIZU International Symposium on Parallel Algorithms/Architecture Synthesis, March 1995.

[7] Jyh-Chang Ueng. "Cohesion: A Distributed Shared Memory System for Multi-computer Network." Master Thesis, Department of Electrical Engineering, National Cheng-Kung Univ., 1993.

[8] Kai Li. "Shared Virtual Memory on Loosely Coupled Multiprocessors." Ph. D. Dissertation, Yale University, October 1986.

[9] Lionel M. Ni, Chong-Wei Xu, and Thomas B. Gendreau. "A Distributed Drafting Algorithm for Load Balancing." IEEE Transactions on Software Engineering, vol. SE-11, no. 11, Oct. 1985.

[10] Phillip Krueger and Niranjan G. Shivaratri. "Adaptive Location Policies for Global Scheduling." IEEE Transactions on Software Engineering, vol. 20, no. 6, June 1994.

[11] Shivaritri N. S., Krueger P. and Singhal M. "Load Distributing for Locally Distributed Systems." IEEE Computer, vol. 25, pp. 33-44, Dec. 1992.

[12] Thomas Kunz. "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme". IEEE Transactions on Software Engineering, vol. 17, no. 7, July 1991.