네트워크 기반 클러스터 시스템을 위한 적응형 동적 부하균등 방법

(Adaptive Dynamic Load Balancing Strategies for Network-based Cluster Systems)

정 훈 진 † 정 진 하 †† 최 상 방 †††

(Hoon Jin Jung) (Jin Ha Jung) (Sang Bang Choi)

요 약 클러스터 시스템은 계산능력과 메모리 크기에 있어서 바람직한 확장성을 제공한다. 또한 고속의 컴퓨터 네트워크 기술의 발달로 인해 클러스터 시스템은 값비싼 MPPs (Massively Parallel Processors)와 비교하여 경쟁력을 얻고 있다. 부적당한 작업 스케줄링은 시스템의 기능을 충분히 이용할 수 없고, 병렬화로부터 얻을 수 있는 이득을 감소시킬 수 있으므로 균등한 작업 분배는 매우 중요한 이슈이다. 병렬처리 프로그램에서 프로그램 실행 전에 각 태스크의 부하를 예측하기는 어려우며, 태스크들은 다양한 형태로 서로 의존적이다. 동적 부하균등 알고리즘에서는 실행시간에 각 프로세서의 부하를 평가한 후, 각 태스크를 적절한 크기로 분할하고 그것들을 각 프로세서의 수행능력에 비례하여 클러스터 시스템에 할당한다. 그러나, 프로세싱 노드간의 통신비용이 높으면, 모든 노드들이 부하분산에 참여하는 것은 효율적이지 못하다. 본 논문에서는 부하분산에 참여하는 프로세서를 통신비용과 평균 부하로부터의 편차를 고려하여 제한하였다. 기존의 부하균등 방식과 제안된 동적 알고리즘을 비교하기 위하여, 통신비용, 노드 수, 그리고 부하의 범위와 같은 파라메터를 사용하여 다양한 모델의 클러스터 시스템에 관한 시뮬레이션을 수행하였다.

Abstract Cluster system provides attractive scalability in terms of computation power and memory size. With the advances in high speed computer network technology, cluster systems are becoming increasingly competitive compared to expensive MPPs (massively parallel processors). Load balancing is very important issue since an inappropriate scheduling of tasks cannot exploit the true potential of the system and can offset the gain from parallelization. In parallel processing program, it is difficult to predict the load of each task before running the program. Furthermore, tasks are interdependent each other in many ways. The dynamic load balancing algorithm, which evaluates each processor's load in runtime, partitions each task into the appropriate granularity and assigns them to processors in proportion to their performance in cluster systems. However, if the communication cost between processing nodes is expensive, it is not efficient for all nodes to attend load balancing process. In this paper, we restrict a processor that attend load balancing by the communication cost and the deviation of its load from the average. We simulate various models of the cluster system with parameters such as communication cost, node number, and range of workload value to compare existing load balancing methods with the proposed dynamic algorithms.

1. Introduction

† 비 회 원 : 일신방사선엔지니어링(주) 연구원
woosuwar@hitel.net
†† 비 회 원 : 인하대학교 전자공학과
g2001131@inhavision.inha.ac.kr
††† 종신회원 : 인하대학교 전자전기컴퓨터공학부 교수
sangbang@inha.ac.kr
논문접수 : 2001년 2월 23일
심사완료 : 2001년 8월 28일

Since the early 1990s, there has been an increasing trend to move away from expensive and specialized proprietary parallel supercomputers toward networks of workstations. The driving forces that made this transition possible are the availability of commodity high performance microprocessors and low-latency networks. These

technologies are making networks of computers an appealing alternative for parallel processing, and this is consequently leading to low-cost commodity supercomputing.

Cluster technology permits organizations to boost their processing power using standard technology (commodity hardware and software components) that can be acquired at a relative low cost. An important factor that has made the usage of cluster system a practical proposition is the standardization of many tools and utilities used by parallel applications. The cluster system is a new supercomputing architecture that have advantages such as high performance, high availability, high scalability and high throughput. The use of cluster systems is becoming increasingly popular since we can build a platform for a given budget which is suitable for a large class of applications and workloads.

Some examples of the cluster system are NOW (Network of Workstations), Beowulf, HPVM (High Performance Virtual Machine), Solaris MC, and so on[1]. Such systems use SPMD(Single Program Multiple Data) style of programming, in which all processors execute copies of the same program for different data space. This does not mean that they proceed in lockstep or even execute the same instructions since, in general, they may follow different control path through the code. Popular message-passing libraries, such as MPI(Message Passing Interface) and PVM(Parallel Virtual Machine), supports SPMD programming paradigm[2].

The cluster system is in position between MPPs (Massively Parallel Processors) and distributed systems. The number of nodes in most cluster systems is around 100, and these nodes communicate each other over high-speed networks using a standard network protocol such as TCP/IP or a low-level protocol such as Active Messages. Thus, the network is the most critical part of a cluster and its capabilities directly influences the performance of the whole system. Cluster systems usually use Fast Ethernet for low cost communication, but sometimes use ATM, Gigabit Ethernet, Myrinet, or SCI network equipment for high performance with an additional cost.

Decomposing a program into multiple tasks usually means that communication will be required among tasks. If these tasks are assigned to different processors, we incur communication among processors through an interconnection network. The primary performance goals of assignment are to balance the workload among processors to reduce the amount of interprocessor communication and the run-time overhead of managing the assignment. If the assignment is completely determined at the beginning of the program, or just after reading and analysing the input, and does not change thereafter, it is called a static or predetermined assignment. Static assignment is effective when the computational requirements of a program are known a prior and do not change during the course of program execution. Whereas, if the assignment of tasks to processor is determined at run time as the program executes to react against load imbalance, it is called a dynamic assignment. The dynamic assignment is an appropriate approach for the program whose workload changes during the execution time, which will necessitate the redistribution of the data in order to maintain the load balance[3].

Although dynamic techniques generally provide good load balancing despite unpredictability of the workload or environmental conditions, they make the management of parallelism more expensive. If the communication cost between processing nodes is high, it is not efficient for all nodes to attend load balancing process because of additional traffic incurred by task or data migration. In this paper, we proposed adaptive dynamic load balancing algorithm, in which processors that participate in load balancing is dynamically restricted by the communication cost and the deviation of its load from the average. During the time interval that there is a wide variation of load among nodes, the proposed algorithm provides coarse tuning in load balancing process. However, when the variation of load becomes small, the algorithm generates fine

tuning in load balancing. We simulate various models of the cluster system with parameters such as communication cost, node number, and range of workload value to compare existing load balancing methods with the proposed dynamic algorithms.

Simulation results show that the proposed algorithms provide better performance than existing algorithms with 5% to 20% speedup in the program execution time. The strong advantage of the proposed adaptive algorithms is the dramatic reduction in the number of migrated tasks, which greatly lessen the network traffic. Especially, our proposed algorithms are very efficient in the network of irregular topology.

The rest of the paper is organized as follows. In Section 2, we classify dynamic load balancing techniques for cluster system and discuss related researches about load balancing techniques. In Section 3, we propose novel dynamic load balancing algorithm and explain the parameters that affect the performance of the cluster system in detail. Then, the simulation results and analyses are represented in Section 4. Finally, concluding remarks are given in Section 5 to summarize this paper.

2. Load Balancing in Cluster System

2.1 Dynamic Load Balancing Strategies

The cluster system is a type of parallel or distributed system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource. A computer node can be a single or multiprocessor system (PCs, workstations, or SMPs) with memory, I/O facilities, and an operating system. In its simplest form, balancing the workload means ensuring that every computer does the same amount of work. In general, load balancing strategies minimize total execution time of an application program under unexpectably varying circumstances through task assignment. They are classified into static load balancing algorithms and dynamic load balancing algorithms[4]. A key issue in exploiting concurrency is whether a good load balance can be obtained by a static or dynamic assignment.

A static assignment is typically an algorithmic mapping of tasks to processors and completely determined at the beginning of the program. Since the assignment is predetermined, static techniques do not incur much task management overhead at run time. However, to achieve good load balance, they require that relative amount of work in different tasks be adequately predictable. The workload to be balanced includes computation and communication. Thus, it is very difficult to predict the load of each task before running the program in network computing environment. In addition to the program itself, environment conditions, such as interferences from other applications, perturb the relationships among processors, thus limiting the robustness of static load balancing.

The dynamic load balancing algorithm, which evaluates each processor's workload in runtime, partitions each task into the appropriate granularity and assigns them to processing nodes in proportion to their performance, thereby minimizing the execution time of the application. After the initial assignment, the dynamic balancing monitors processor performance, calculates new distribution, and then moves the data between processors. The performance of dynamic load balancing algorithms heavily depends on the accuracy of each balancing decision and the amount of added processing and communication incurred by the redistribution.

Dynamic load balancing strategies are either global or local, based on the information they use to make load balancing decision[5]. In the global scheme, all the processors take part in the synchronization and send their performance profile to the load balancer. Thus, the load balancer can make an optimal workload balancing decision until that point using global knowledge. However, communication or synchronization cost is more expensive. In the local scheme, we partition the processors into several groups, where the groups can remain fixed for the duration of execution or the membership can be changed dynamically. And the load balancing decisions are made only within a

group. Thus, workload balancing is not optimal and converges more slowly. However, communication or synchronization cost is cheaper.

We can also classify dynamic load balancing techniques into centralized or distributed, depending on whether the load balancer is located at a master processor or distributed among the processors, respectively[1]. In the centralized techniques, one balancer could prevent the cluster system from scaling up to a larger system. On the other hand, the distributed schemes require synchronization that involve all-to-all broadcast.

In previous researches, loop iterations are mapped to virtual processors, and these virtual processors are assigned to the physical processors based on past load behavior[6]. Loop scheduling algorithm called affinity scheduling is also proposed in[7]. This algorithm attempts to balance the workload, minimize synchronization operation, and exploit processor affinity. A global centralized scheme and a local distribution scheme are implemented in Dome[8], where the performance metric used is the rate at which processors execute the Dome program, and load balancing involves periodic exchanges. Dynamic load balancing scheme for switch-based network of workstations is discussed in[9]. The proposed method uses global information to achieve an accurate load balancing and all processing elements cooperate to broadcast or collect load information to reduce the communi · cation overhead. Static scheduling algorithms for heterogeneous programs, processors, memory, and work were proposed in[10]. There has been relatively little work in static scheduling in heterogeneous clusters.

If the communication cost between processing nodes is low, it is efficient for all nodes to attend load balancing process. However, if the communication cost becomes expensive, it is not desirable for all nodes to be involved in the load balancing due to additional traffic incurred by task or data migration. In this paper, proposed dynamic load balancing algorithm estimates load imbalance among processors and invoke the load balancer only for those processors for which the potential speedup obtainable through balanced workload is greater than the balancing overhead. We measure the load imbalance with the communication cost and the deviation of its load from the average.

2.2 Proposed Load Balancing Model

Cluster systems efficiently supports SPMD programming paradigm. A large number of parallel programs belongs to this class. For example, linear algebra problems, partial differential equation solvers, and image processing algorithms can be easily structured into SPMD program[2]. Since the workload on each processing node is a function of the number of data elements assigned to it and communication cost incurred by the tasks. In its simplest form, we can keep the balanced workload by means of data migration from overloaded to underloaded nodes.

The dynamic load balancing algorithm proposed in this paper uses several parameters to estimate workload value for each processing node. These values are used as inputs to the load balancer to detect load imbalances and make a load migration decision. The parameters include the number of tasks in ready queue, CPU utilization, and I/O usage. As the program execution progresses, the inaccuracy of the workload estimate leads to unbalanced load distributions. The imbalance should be detected, and then the balancer has to perform an appropriate migration strategy to correct the imbalance.

Figure 1 shows a distribution of workload according to the estimated load of processing nodes in a cluster system. The maximal load is defined as the load value that the most heavily loaded node has. And the minimal load is the load value that the most underloaded node has. We also define average load as the overall workload of nodes divided by the number of nodes. The upper margin α is defined as a fraction of the distance from the average to the maximal load, and the nodes between *average* and *average* + (*maximal* - *average*) * α workload are called as lightly overloaded nodes. In a similar manner, we can

define lower margin β and the nodes between *average* + (*minimal* - *average*) * β and *average* are referred to as lightly underloaded nodes. The upper margin (α) and lower margin (β) are adaptively determined considering current communication cost and load balancing overhead. Notice that α and β are not load values, but they are deviation ratios from an average load value. If we assume that average load value is 100, maximal load value is 150, and α is 10%, then a node with load value 114 is considered as an overloaded node. Whereas, if the maximal load value is 180, a node with the same load value is regarded as a lightly overloaded node. Table 1 shows the classification of nodes in accordance with relative workload.

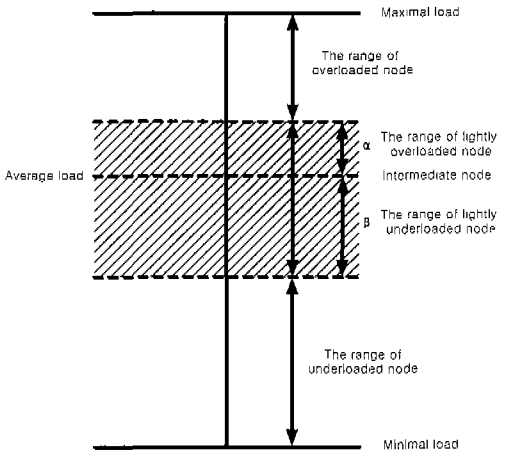

Fig. 1 Distribution of workload

Table 1 Classification of processing nodes

Classification of nodes	Workload Range
overloaded node	$\chi > Average + (Max - Average) * \alpha$
lightly overloaded node	$Average < \chi \leq Average + (Max - Average) * \alpha$
intermediate node	$Average$
lightly underloaded node	$Average - (Average - Min) * \beta \leq \chi < Average$
underloaded node	$\chi < Average + (Min - Average) * \beta$

Average : Average load value
Max : Maximal load value
Min : Minimal load value
χ : Workload value of a node

Load balancing process we are using in this paper proceeds in five phases as follows.

(1) Synchronization: In our algorithm, a synchronization is triggered by a master node that is selected among underloaded nodes. Then each node sends its performance profile to the load balancer in the master node.

(2) Processor load estimation: A workload value is calculated for each processor in the cluster system. We try to predict the workload value based on the past history (CPU utilization and I/O usage) of the processor and the remaining load (number of tasks in ready queue).

(3) Load balancing profitability analysis: The load balancer quantifies the degree of load imbalance according to Figure 1 for each processor. We use estimated workload values for the classification of nodes considering the load balancing overhead and communication cost.

(4) Task migration decision: Sources and destinations for task migration are determined using estimated workload value based on the network topology. Source nodes are notified of the quantity and destination node of tasks for load balancing. In our approach, only overloaded and underloaded nodes participate in load balancing process.

(5) Task migration: Instructed source nodes determine adequate tasks for efficient load balancing and send them to the designated destination nodes.

The degree of global knowledge used in the balancing process is critical to the accuracy of balancing decision. Centralized approaches are more accurate since the entire systems's state information is gathered to a single master node and accurately predicted workload values are used in the decision process. However, the collection of performance profile from all nodes may require synchronization which incurs an overhead and a

delay. In our approach, this overhead can be offset by limiting the nodes that participate in the load balancing process. Decentralized or local approaches are less accurate since they decide task migration based on the information only within a group of nodes. But they incur a smaller synchronization overhead. In this paper, we simulate both of the schemes to compare the performance.

3. Adaptive Dynamic Load Balancing

The cluster systems considered in this paper consist of N homogeneous nodes connected by switch-based network. Master node is selected out randomly among underloaded nodes. The master node has to perform an additional job to collect performance profile from the other processors and distribute load. We assume that the communication of the switch-based network is carried out with one-port model. The one-port model restricts a node to exchange messages with at most one node at a time. We also assume that multicast is not supported in hardware.

The dynamic load balancing algorithm proposed in this paper can be explained with the following four policies[11].

(1) Information policy: It decides when the state of the other processors is to be gathered. The master node periodically receives load information updates from all other nodes in the proposed algorithm.

(2) Transfer policy: It determines whether a node needs to participate in a task transfer, either as a sender or receiver. In this paper, only overloaded nodes between *maximal* and *average* + (*maximal* - *average*) * α send extra load. And only underloaded nodes between *minimal* and *average* + (*minimal* - *average*) * β receive the extra load.

(3) Selection policy: It selects a task for transfer. In this paper, a task is eligible for migration only if it is in the ready queue, its execution time is greater than the migration overhead, and it is not a task that was transferred from other nodes.

(4) Location policy: Its responsibility is to find suitable migration partners. An underloaded node receives a task from one or more overloaded nodes in the near neighborhood in the proposed algorithm.

In the proposed dynamic algorithm, the following 5 steps are performed in each load balancing process.

Step 1. Synchronization: A master node broadcasts a message indicating the start of load balancing to collect information from the other nodes. The communication cost of synchronization step is $C_{sync} = C_{msg}$, where C_{msg} represents the cost for sending or receiving synchronization or load information.

Step 2. Load estimation: Each slave node calculates its workload value based on the past history and number of tasks in ready queue. Then it sends estimated workload value to a master node. The communication cost of processor load estimation step is $C_{receive} = (N - 1) \times C_{msg}$, where N is the number of nodes in a given cluster system.

Step 3. Profitability analysis: The load balancer in the master node calculates workload distribution based on the received load information. For each processor, it classifies the degree of load imbalance in accordance with Table 1. The load balancer determines α and β based on communication cost and load balancing overhead. Hence, the proposed load balancing algorithms are referred to as adaptive. Two lists, overloaded node list and underloaded node list, are created. And the overloaded and underloaded node lists are sorted in decreasing and increasing order, respectively, according to workload value.

Step 4. Migration decision: Sources and destinations for task migration are determined using two lists obtained from the previous step and distance between nodes based on the given network topology. Basically, the node with the lowest load tends to receive tasks from a node with the highest load in local domain. The load balancer notifies each source node of the amount of tasks and destination node. Underloaded nodes are also notified of the migration decision. The lightly overloaded nodes and lightly underloaded nodes do

not participate in load balancing process. The communication cost of task migration decision is $C_{decision} = (N - 1 - M) \times C_{msg}$, where M is the number of lightly overloaded or underloaded nodes.

Step 5. Task migration: Each source node picks out suitable tasks according to the selection policy and sends them designated destination node determined in the previous step according to the location policy. The cost of task migration is approximately proportional to the number of tasks. Thus, $C_{task} = C_{msg} \times$ *number of tasks*. In order to synchronize, two additional C_{msg} costs are also added. Then, the communication cost of task migration is $C_{mig} = 2C_{msg} + C_{task}$.

In the proposed algorithm, a node with load value χ is considered as lightly overloaded node if χ is a value between *average* and *Average* + (*Max* - *Average*) * α. However, when there is a small variation in the load distribution among nodes, χ can be larger than *Average* + (*Max* - *Average*) * α. In the circumstance, a node with the same load value will be considered as overloaded node and take part in load balancing. In other words, when there is a small variation in the load distribution, the proposed algorithm performs fine tuning in load balancing. However, during the time period that there is a wide variation of load, fine tuning does not give profit commensurate with load balancing cost. It is the period that generates large imbalance of load among nodes and there is a high probability that the balanced load will be disturbed again soon. Thus, the proposed algorithm performs coarse tuning of load when there is a wide variation of load among nodes. If the variation of load becomes small, then the algorithm generates fine tuning in load balancing.

4. Simulation and Analysis

4.1 Simulation and Load Model

In this paper, we studied dynamic load balancing using a simulation model. We compared two generic dynamic load balancing algorithms, centralized dynamic load balancing (CLB) and decentralized load balancing (DLB)[10], with the proposed adaptive central dynamic load balancing (ACLB) and adaptive decentralized dynamic load balancing (ADLB). As explained in the preceding section, proposed adaptive algorithms dynamically restrict the processors that take part in load balancing process considering the communication cost and load distribution. For higher communication cost, larger values of α and β are used in the profitability analysis. Table 2 shows simulation parameters for dynamic load balancing algorithms.

Most cluster systems employ switch-based networks and they are composed of several groups of nodes. Generally, the number of nodes in each group is a multiple of 2. We constitute cluster systems with two types of groups. The cluster system with irregular groups has different number of nodes in each group. Whereas the system with regular groups has the same or similar number of nodes in each group. Figure 2 shows an example of the cluster system configured with irregular groups, where five groups consist of 1, 2, 4, 8 and 16 nodes, respectively. Figure 3 shows a cluster system with regular groups, where each group consists of the same number of nodes, i.e., six nodes.

Table 2 Simulation parameters for proposed dynamic load balancing algorithm

Parameters	Meaning	Used values
T_{period}	load balancing period	1000 unit time
T_{total}	total number of tasks in a program	30000 unit time
C_{msg}	communication cost to send load information	1 ~ 5% of *task execution time*
$C_{receive}$	communication cost to receive entire load information	$(N - 1) \times C_{msg}$
C_{task}	cost of task migration	$C_{msg} \times$ *number of tasks*
C_{mig}	communication cost for task migration	$2C_{msg} + C_{task}$
α, β	upper, lower margin of workload distribution	10 - 40%
N	number of nodes in a cluster system	31, 63, 127 nodes
M	number of lightly loaded nodes	d % of N nodes
S	switch overhead	$C_{msg} \times 1.4$

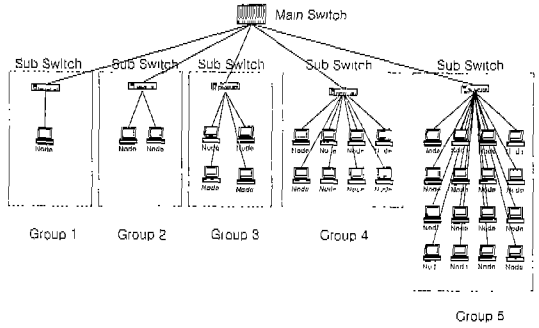

Fig. 2 An example of network configuration with irregular groups

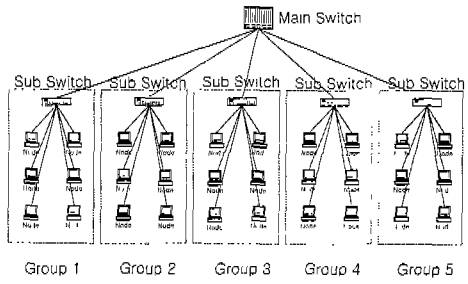

Fig. 3 An example of network configuration with regular groups

Synthetic benchmark programs are practical for the analysis of a load balancing algorithm since both the size and quantity of the tasks being processed can be predetermined. We assumed that a parallel application program is partitioned into independent tasks and the amount of job that each task should perform has a uniform random distribution. It is a fundamental assumption in most load balancing algorithms. Even if the same number of tasks is initially distributed among nodes, it is impossible to predict the exact execution time because each task has variable amount of jobs to do. The communication cost for C_{msg} is assumed to be one unit time between two nodes in the same group. And we suppose that a message C_{msg} that goes through main switch takes 1.4 unit time.

In the load model of our simulation, 100 tasks are initially assigned to each processor. Each task has the following distribution for its execution time. Execution time of a task j in node i is denoted as $\tau_j(i)$. And $\tau_j(i)$ has an uniform random distribution in range $0 < \tau_j(i) < 2\tau(i)$, where $\tau(i)$ is an average execution time of all tasks in node i. $\tau(i)$ is again randomly selected in range $0 < \tau(i) < 2E(\tau)$, where $E(\tau)$ is an average execution time of all tasks for a given application. We vary C_{msg} from 1% to 5% of $E(\tau)$. The task migration time C_{task} is set to C_{msg} times the number of tasks.

4.2 Simulation Results and Analyses

Figures 4 through 8 show simulation results for a cluster system with irregular groups. Speedup is the ratio of total execution time of the system with no balancing to that of the system with load balancing.

$$Speedup = \frac{T_{NoLB}}{T_{LB}}$$

In the simulation, α does not always equal to β since the sum of surplus load in overloaded nodes must be equal to the whole deficient load of underloaded nodes. Figure 4 shows that if the communication cost is less than 4% of $E(\tau)$, the performance of the centralized algorithms (CLB, ACLB) is better than that of the decentralized algorithms (DLB, ADLB) since communication overhead is relatively small. However, decentralized algorithms are insensitive to communication overhead. Thus, if the communication cost becomes more than 4% of $E(\tau)$, the decentralized algorithms are superior to the centralized algorithms. In either cases, proposed adaptive dynamic algorithms show excellent performance compared to the corresponding generic algorithms. Figure 5 depicts speedup obtained from the dynamic balancing algorithms for various values of α. This graph shows that proposed algorithms show better performance when α is 20% to 30%. It is believed that there are no direct methods to obtain optimal choice for the value of α since it is a complex function of various variables related to network traffic and load distribution. Analytical method to choose optimal value of α will be performed as a future research.

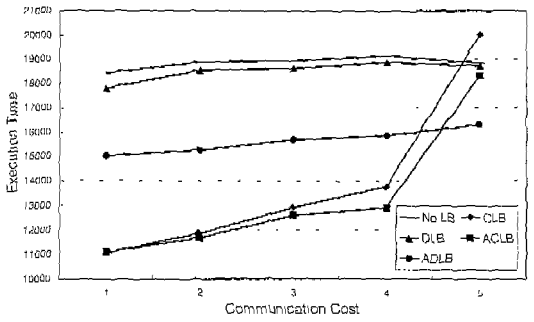

Fig. 4 Execution time versus communication cost for α=30%, 31 nodes, and irregular groups

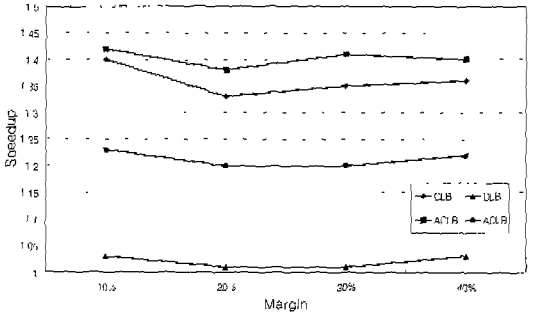

Fig. 5 Speedup of different margins (α) for 31 nodes and irregular groups

Figures 6 and 7 reveal the number of migrated tasks that each dynamic algorithms induce to distribute load for various values of α and communication cost, respectively. For larger values of α, smaller number of tasks are migrated. CLB transfers the largest number of tasks to balance the load among processors. The strong advantage of the proposed adaptive algorithms is the dramatic reduction in the number of migrated tasks, which greatly lessen the network traffic. The performance that Figure 8 shows is measured in terms of speedup for three different sizes of cluster systems. If a cluster system consists of 31 or 63 nodes, ACLB shows the best performance since the centralized algorithm can achieve good load balancing because of the nonuniform number of nodes in each group. However, in a cluster system with 127 nodes, ADLB shows the best performance because each group consists of large number of nodes even though they are not regular, and decentralized algorithm can achieves fairly good balancing with low overhead.

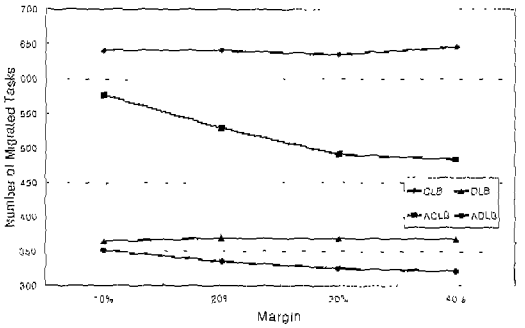

Fig. 6 Number of migrated tasks versus various margins (α) for 31 nodes and irregular groups

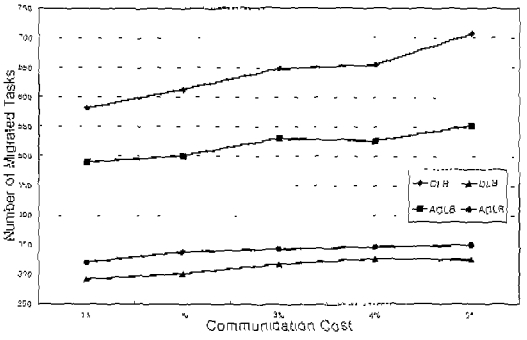

Fig. 7 Number of migrated tasks versus communication cost for 31 nodes and irregular groups

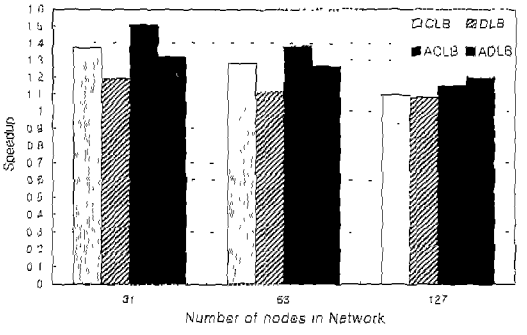

Fig. 8 Execution time of different size systems with irregular groups

Figures 9 through 13 represent simulation results for a cluster system with regular groups. Figure 9 shows that the performance of the centralized algorithms is better than that of the decentralized algorithms for small communication overhead, but the former becomes inferior to the latter when the communication cost is larger than 4% of $E(\tau)$. The difference of execution times between the centralized and decentralized algorithms are not dramatic compared with Figure 4, since each regular group has a sufficient number of processors to balance the load within its own group.

Figure 10 represents the speedup in program execution time obtained from the dynamic load balancing algorithms for α from 10% to 40%. This graph shows that DLB provides better performance than ADLB in contrast with Figure 5, since each regular group has a sufficient number of processors to balance the load within the group. If there are many processors in each group, we can have a high probability to achieve a good load balance. Furthermore, the communication cost is low within a group. As a result, DLB can accomplish fairly good performance.

Figure 11 depicts the number of migrated tasks that each dynamic algorithms induce to distribute load for various values of α. In a cluster system with irregular group, some groups consist of very small number of nodes. Thus, only small number of tasks can be transferred within such groups by decentralized algorithms as shown in Figure 6. On the other hand, the system with regular groups, which is employed in Figure 11, has 6 or 7 nodes in each group and DLB transfers larger number of tasks than adaptive algorithms. Figure 12 shows that larger number of migrated tasks is obtained for larger communication cost to balance the load. For large communication cost, the overhead to

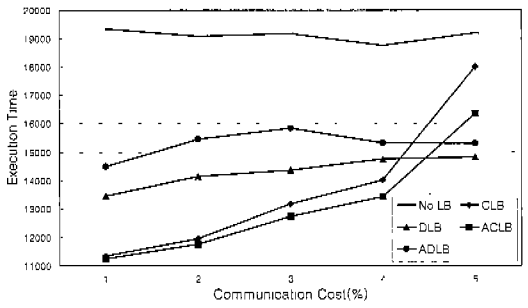

Fig. 9 Execution time versus communication cost for α = 30%, 31 nodes, and regular groups

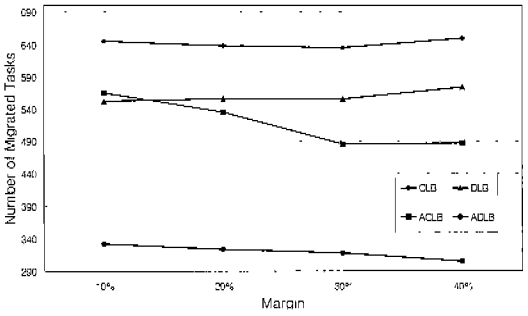

Fig. 11 Number of migrated tasks versus various margins (α) for 31 nodes and regular groups

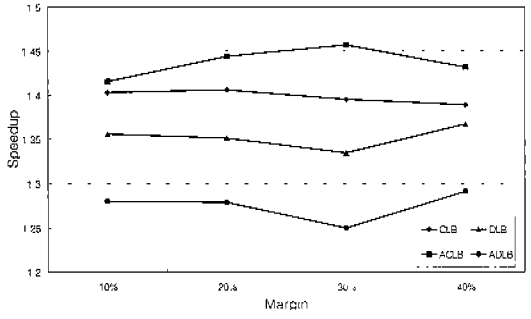

Fig. 10 Speedup by different margins (α) for 31 nodes and regular groups

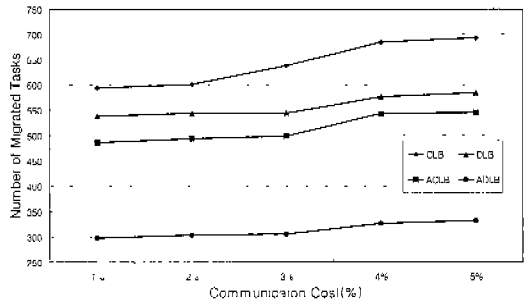

Fig. 12 Number of migrated tasks versus communication cost for 31 nodes and regular groups

achieve the load balancing will be excessive. Thus CLB and DLB transfer huge amount of tasks to distribute load, but also lead to performance degradation due to the excessive overhead and network traffic, which can offsets the performance gain obtained form the load balancing

Figure 13 represents the performance gain measured in terms of speedup for three different sizes of cluster systems with regular groups. Figures 6 and 13 reveals the fact that the cluster system with regular groups provides better performance than the system with irregular groups since nodes are uniformly distributed among groups in the former. Regardless of the regularity of network, for large number of nodes, relatively low speedup is obtained for all algorithms. It is assumed that the communication cost is from 1% to 5%. It is a fairly big overhead since large number of tasks are migrated due to the load imbalance that the simulator artificially creates. As a result, the efficiency of load balancing is decreased by the communication overhead.

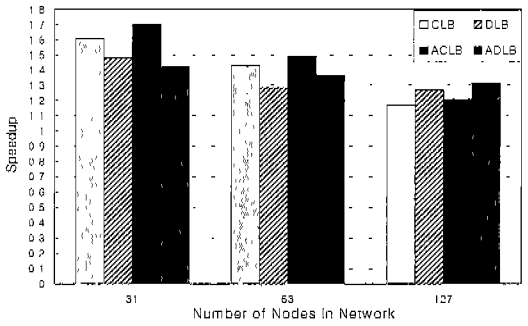

Fig. 13 Execution time of different size systems with regular groups

5. Conclusion

The cluster system usually has irregular network architecture connected with high-speed switch. Thus, we need a different approach for load balancing as compared with load balancing used in MPPs with very regular interconnection network. We propose two adaptive load balancing algorithms which dynamically limit processors that take part in load balancing process according to the distribution of load among nodes in the cluster system. We simulate two types of network topology, i.e., cluster system with regular group and cluster system with irregular group. In the simulation model, we include various architecture parameters such as communication cost, node number, and range of workload values to compare existing load balancing methods with the proposed adaptive algorithms.

Simulation results show that proposed ACLB and ADLB provide better performance than existing algorithms with 5% to 20% speedup in the program execution time. CLB and DLB transfer huge amount of tasks to distribute load, but also lead to performance degradation due to the excessive overhead and network traffic. For high communication cost, the overhead to achieve the load balancing will be excessive, which can offsets the performance gain obtained from the load balancing by these two algorithms. The benefit obtained from the proposed two adaptive algorithms is the dramatic reduction in the amount of migrated tasks, which greatly reduces the network traffic. Especially, the proposed algorithms are very efficient in the network of irregular topology.

References

[1] R. Buyya, High performance cluster computing: Architectures and systems, volume 1, Prentice Hall, 1999.

[2] M. Cermele, M. Colajanni, and G. Necci, "Dynamic load balancing of distributed SPMD computations with explicit message-passing," Sixth Heterogeneous Computing Workshop, pp. 2-16, Apr. 1997.

[3] M.H. Willebeek-LeMair and A.P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 9, pp. 979-993, Sep. 1993.

[4] D.E. Culler and J.P. Singh, Parallel computer architecture: A hardware/software approach, Morgan Kaufmann Publishers, 1999.

[5] M.J. Zaki, W. Li, and S. Parthasarathy.

"Customized dynamic load balancing for a network of workstations," Journal of Parallel and Distributed Computing, vol. 43, no. 2, pp. 156-162, June 1997.
[6] N. Nedeljkovic and M. Quinn, "Data-parallel programming on a heterogeneous workstations," First International Symposium on High-Performance Distributed Computing, pp. 28-36, Sep. 1992.
[7] E.P. Markatos and T.J. LeBlanc, "Using processor affinity in loop scheduling on shared-memory multiprocessors," IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 5, pp. 379-400, Apr. 1994.
[8] J. Arabe et al., "Dome: Parallel programming in a heterogeneous multi-user environment," Technical Report 95-137, Carnegie Mellon University, Apr. 1995.
[9] W.Y. Lee, S.J. Hong, J. Kim, and S. Lee, "A dynamic load balancing algorithm on switch-based networks," 13th International Conference on Parallel and Distributed Computing Systems, Aug. 2000.
[10] M. Cierniak, M.J. Zaki, and W. Li, "Compile time scheduling algorithms for a heterogeneous NOW," The Computer Journal, vol. 40, no. 6, pp. 356-372, Dec. 1997.
[11] W. Cai, B.S. Lee, A. Heng, and L. Zhu "A simulation study of dynamic load balancing for network-based parallel processing," Third International Symposium on Parallel Architecture, Algorithms, and Networks, pp. 383-389, Dec. 1997.

정 훈 진
1998년 2월 수원대학교 전자공학과(학사). 2000년 8월 인하대학교 전자공학과(석사). 2000년 8월 ~ 현재 일진방사선 엔지니어링(주). 관심분야는 Cluster computing.

정 진 하
1992년 인하대학교 전자공학과(학사). 1994년 인하대학교 전자공학과(석사). 1994년 ~ 1999년 (주)한미 기술연구소. 2000년 인하대학교 전자공학과(박사과정). 관심분야는 컴퓨터구조, Fault-tolerant computing, Parallel processing.

최 상 방
정보과학회논문지 : 시스템 및 이론 제 28 권 제 1 호 참조