Dynamic Load Balancing in Parallel Processing on Heterogeneous System

Su Thet Aung, Khin Kyu Kyu
Computer University, Pathein
ms.suthet@gmail.com, khinkyu28@gmail.com, mayphyooo@gmail.com

Abstract

This paper presents the parallel computing and dynamic load balancing on heterogeneous system architecture, simultaneously analyzing the theoretical parallel Speedup as well as the Speedup experimentally obtained. Load balancing techniques play a very important role in developing high performance system. However, the parallel computing algorithm use that one slave computer calculates one problem. If the problem is small and the processor speed is high, the performance of the system is increased. If the problem is very large but the processor speed is slow, the performance of the system is decreased. Hence, dynamic load balancing system is developed to evaluate performance of load balancing algorithm and to calculate on the slave nodes. The proposed dynamic load balancing scheme can minimize the average slow down of all parallel jobs running on a system and reduces the average response time of jobs.

1. Introduction

A distributed system consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal. Distributed computing also refers to the use of distributed systems to solve computational problems. In distributed computing, a problem is divided into many tasks, each of which is solved by one computer.

Decrease in hardware costs and advances in computer networking technologies have lead to increased interest in the use of large-scale parallel and distributed computing systems. One of the biggest issues in such systems is the development of effective techniques/algorithms for the distribution of the processes/load of a parallel program on multiple hosts to achieve goal(s) such as minimizing execution time, minimizing communication delays, maximizing resource utilization and maximizing throughput. Using queuing analysis and assuming job arrivals in a multi-host system the probability of one of the hosts being idle while other host has multiple jobs queued up can be very high. Such imbalances in system load suggest that performance can be improved by either transferring jobs from the currently heavily loaded hosts to the lightly loaded ones or distributing load evenly/fairly among the hosts. The algorithm is known as load balancing algorithm. [6] Dynamic load balancing algorithms (DLB) are adaptive to changing situations and take decisions at run time.

In this paper, three heterogeneous nodes connect by a switch-based network. The master node can predict the average execution time of tasks for each slave node based on the information from the corresponding slave node. Then, the master node redistributes remaining tasks to each node considering the predicted execution time. Dynamic load balancing uses execution time prediction to optimize the task redistribution.

2. Model Structure

2.1 Network Model

Figure 1 shows master-slave network model. The parallel programs simulated follow the master and slave network model, where a master task generates a number of slave tasks. [2] Each slave task carries out some processing and sends result back to the master. After receiving the results from all the slaves, the master task will be terminated.

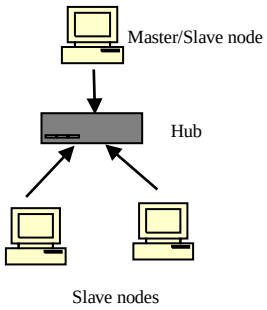

Figure1. Master- Slave Network Model

The system considered in this paper consists of three heterogeneous nodes and have switch-based network. The master node is randomly selected among nodes and has additional jobs which distribute tasks and gather results from slave nodes, processing own task as one node.

2.2 Parallel Computing

Parallel computing algorithm is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be [5] divided into smaller ones, which are then solved concurrently ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data and task parallelism. In this system, task parallelism is a form of parallelization of computer code across multiple processors in parallel computing environments. Task parallelism focuses on distributing execution processes across different parallel computing nodes. In this system, task parallelism is achieved when each processor executes a different process on the same or different data because of using three processors. The process may execute the same or different problem. In the general case, different [3] execution threads communicate with one another as they work. Communication takes place usually to pass data from one thread to the next as part of a workflow.

2.3 Load Balancing

Load balancing on multi computers is a challenge due to the autonomy of the processors and the inter processors communication overhead incurred in the collection of state information, communication delays, redistribution of load, etc. Parallel and distributed computing environment is inherently best choice [1] for solving/running distributed and parallel program applications. In such type of applications, a large process/task is divided and then distributed among multiple hosts for parallel computation. Has pointed out that in a system of multiple hosts the probability of one of the hosts being idle while other host has multiple jobs queued up can be very high. Here load balancing is likely to improve performance. Such imbalances in system load suggest that performance can be improved by either transferring jobs from the currently heavily loaded hosts to the lightly loaded ones or distributing load evenly/fairly among the hosts .The algorithms known as load balancing algorithms, helps to achieve the above said goal(s). [4] The processors are categorized according to workload in their queues as heavily loaded (more tasks are waiting to be executed), lightly loaded (less tasks are waiting to be executed in queue) and idle processors/hosts (having no pending work for execution). Here queue length is used as an indicator of workload at a particular processor. But none of them is found to be an idle.

3. Dynamic Load Balancing Algorithm

1. Nature

DLB algorithms are of dynamic and no-planning nature as tasks are assigned at run-time to processors and tasks redistribution can take place if task assignment that was earlier done is not giving good performance (that is if proper balancing of load is not there). So their behavior is totally nondeterministic and no initial planning is done for assigning load to hosts as this work is done at run-time.

2. Resource Utilization

DLB algorithms have relatively better resource utilization as dynamic load balancing take care of the fact that load should be equally distributed to processors so that no processors should sit idle.

3. Preemptive

DLB algorithms are both preemptive and non preemptive.

4. Predictability

DLB algorithm's behavior is unpredictable, as everything has been done at run time.

5. Adaptability

DLB algorithms are adaptive towards every situation whether numbers of processes are fixed or varying one.

6. Reliability

DLB algorithms are relatively more reliable processes can be transferred to other nodes in case of failure of node occurs.

7. Response Time

DLB algorithms may have relatively higher response time as sometimes redistribution of processes takes place. Sometime is being consumed during task migration.

8. Stability

DLB such kind of information is exchanged among processors and if this information is out of date i.e. information which is not updated regularly

or periodically among processors then it can lead the whole system to an unstable state.

4. The Proposed System

In proposed system, Master/ Slave paradigm use three heterogeneous computers with different calculation powers. The algorithm of the system has two main parts. First, parallel computing algorithm (without load balancing) calculates the matrix operations. Secondly, dynamic load balancing algorithm (with load balancing) also calculates the matrix operation. In both algorithms request IP Address from nodes. And these algorithms accept input data (Matrix operations) with text file. Each text file has many matrix operations with text ID numbers. Parallel computing algorithm and dynamic load balancing algorithm have two main parts (master and slave programs), and use three nodes. One node runs master program and slave program. Two nodes run slave nodes. Master program send the jobs to each slave node and collect the calculated results from slave node. Slave program calculates the problems on each slave nodes. If each slave nodes calculate the tasks complete, send the results from each slave to master node. Figure 2 is the overview of the proposed system is described in Figure 2.

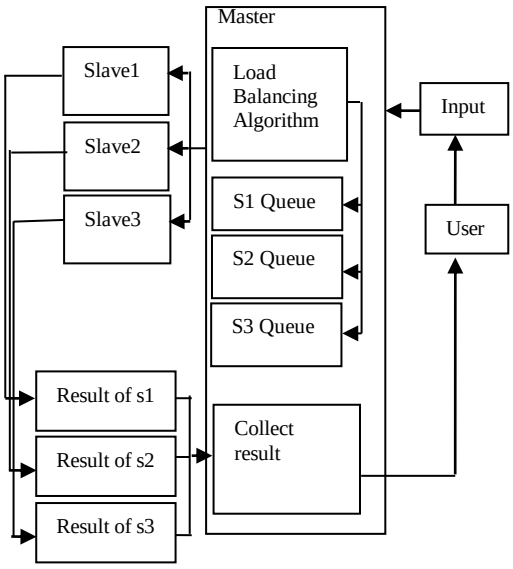

Figure2. System Overview

4.1 Parallel Computing Algorithm (Without Load Balancing)

Parallel computing algorithm calculates many problems on a 3 heterogeneous processors system (CPUs “a”, “b” and “c”) in a parallel environment and to do tasks “A” ,”B” and “C”, and then tell CPU “a” to do task “A”, CPU “b” to do task “B” and CPU “c” to do task “C” simultaneously, thereby reducing the runtime of the execution. Tasks “a”, “b” and “c” have each text file that have many matrix operations problems. The tasks can be assigned using conditional statements as following algorithm.

```
program:
...
if CPU="a" then
do task "A"
else if CPU="b" then
do task "B"
else CPU="c" then
do task "C"
end if
...
end program
```

The following Figure 3 shows the processing of Parallel computing Algorithm

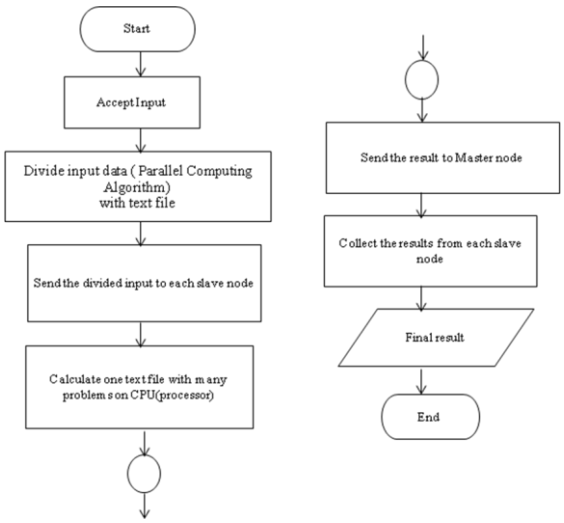

Figure3. System Flow Diagram of Parallel Processing

4.2 Dynamic Load Balancing Algorithm

Dynamic load balancing algorithm calculates many problems on 3 heterogeneous processors. Dynamic load balancing algorithm where a percentage of the total workload will be allocated to the processor at the moment of starting the application, according to each processor will demand more work on the part of the Master, as its task is being completed. In dynamic load balancing

algorithm, master program monitors the tasks and queue length, divide the problems, and collects the results. If the master program divides the problems, it gives to each slave with tasks ID numbers in each text file. Slave program calculates the matrix operations. When one slave nodes is free, the master program re-divides the tasks.

find out if I am MASTER or SLAVE

```
if I am MASTER
    do until no more jobs
       send to SLAVE next job
       receive results from SLAVE
     end do
     tell SLAVE no more jobs

else if I am SLAVE
     do until no more jobs
       receive from MASTER next job
 calculate jobs until the jobs has been completed
       send results to MASTER
      end do
endif
```

The dynamic load balancing using predictable execution time without consideration of communication cost. In order to achieve the dynamic load balancing, master node has to calculate a task to move using the load information of slave node. N is the number of task that distributed with the nodes of the number of *P* equally, master node per time t_j collects the information of task number (n_i) in ready queue of each slave node i , 1, …, P. Master node calculates average time that each slave node executes a task through the collected information during period t_j.

$$T_{task_i} = \frac{t_j}{N - n} \quad , i = 1 \ldots P \quad \ldots\ldots\ldots\ldots\ldots (1)$$

N_{total} is the total number of tasks on slave node, then

$$N_{total} = \sum_{i=1}^{n_j} \quad \ldots\ldots\ldots\ldots\ldots (2)$$

Ideally, in order to get the best performance of cluster system, all slave nodes should finish the task simultaneously. Thus, as shown in the following equation, all slave nodes should be redistributed by the task in the ratio of the performance of each slave node.

$$n'_1 \times T_{task_1} = n'_2 \times T_{task_2} = \ldots = n'_p \times T_{task_p}$$

$\ldots\ldots\ldots\ldots\ldots\ldots$ (3)

Solving the above two equations (i.e. equations 2 & 3), we can obtain the following formula for n'$_i$:

$$n'_i = \frac{1}{T_{task_i} \sum_{j=1}^{P} \frac{1}{T_{task_j}}} \times N_{total} \ , \ i = 1 \ldots P$$

$\ldots\ldots\ldots\ldots\ldots\ldots$ (4)

DLB algorithms that consider the current load conditions (i.e. at execution time) in making job transfer decisions. DLB algorithm can be redistributed workload among hosts at the runtime as the circumstances changes i.e. transferring the tasks from heavily loaded processors to the lightly loaded ones. Dynamic load balancing algorithm continually monitor the load on all the processors, and when the load imbalance reaches some predefined level, this redistribution of work takes place. The system adopts the number of tasks in the ready queue of each node for measuring the load. The processors are categorized according to workload in their queues as heavily loaded (more tasks are waiting to be executed), lightly loaded (less tasks are waiting to be executed in queue) and idle processors/hosts (having no pending work for execution). Here queue length is used as an indicator of workload at a particular processor. But none of them is found to be an idle.

A DLB algorithm considers following issues:

(1) Load estimation policy, which determines how to estimate the workload of a particular node of the system.

(2) Process transfer policy, which determines whether to execute a process locally or remotely.

(3) State information exchange policy, which determines how to exchange the system load information among the nodes.

(4) Priority assignment policy, which determines the priority of execution of local and remote processes at a particular node.

(5) Migration limiting policy, which determines the total number of times a process, can migrate from one node to another.

The system flow diagram of dynamic load balancing algorithm as shown in below:

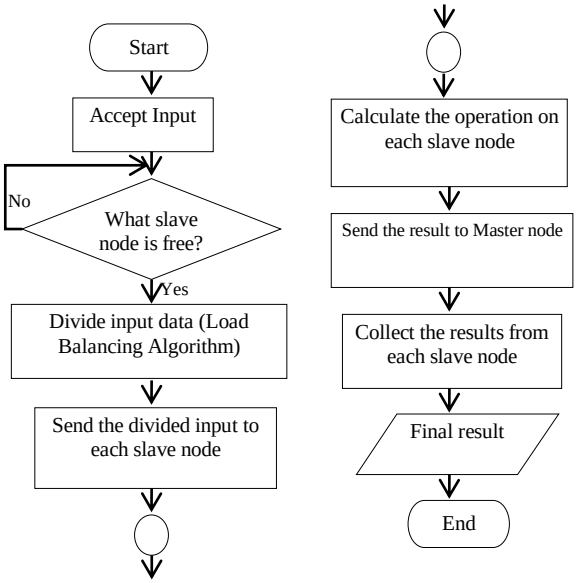

Figure4. System Flow Diagram of Dynamic Load Balancing

4.3 System result

Parallel computing algorithm and dynamic load balancing algorithm calculate the matrix operations. Parallel computing algorithm calculates one problem at one processor. If the problem is small and the processor speed is fast, the performance of the system is increased. If the problem is very large but the processor speed is slow, the performance of the system is decreased. Hence, this system can overcome those above problems and can be adaptive to calculate with slave nodes using dynamic load balancing algorithm. Dynamic load balancing algorithm is adaptive to calculate with salves nodes.

The experimentally results are shown in following figures:

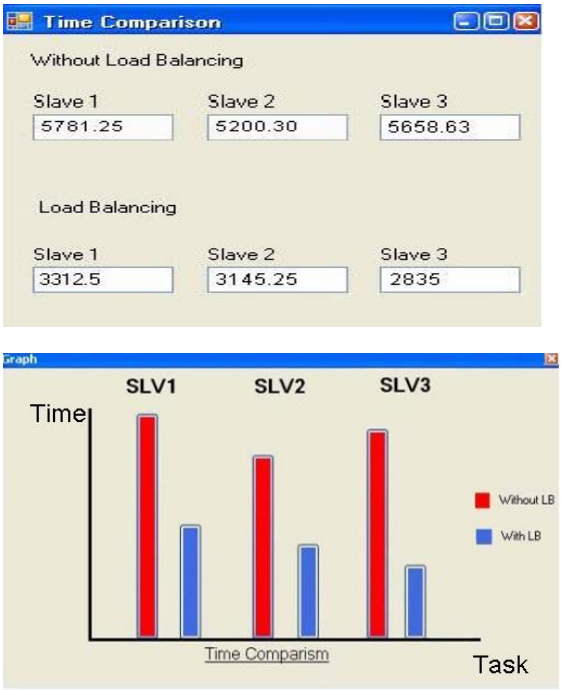

Figure5. Compare of parallel computing (without load balancing) and load balancing

5. Conclusion

In this paper, the proposed dynamic load balancing uses execution time prediction to optimize the task redistribution. The various performance factors such as number of nodes, number of tasks are considered to improve the performance of the system. The benefit of dynamic load balancing algorithm is mapping objects on to a parallel machine. In dynamic distribution, the Speedup obtained is quite close to the optimum according to the parallel architecture used in this system. Hence, the proposed dynamic load balancing can minimize the average slow down of all parallel jobs running on a system and reduces the average response time of jobs.

8. Reference

[1] Y.C. Chow and W. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Mu1tiple Processor System," *IEEE Transactions on Computers*, Vol. C-28, pp. 334-361.

[2] 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.

[3] D L Eager, E D Lazowska , J Zahorjan, “A comparison of receiverinitiated and sender-initiated adaptive load sharing”, *Performance Evaluation*, v.6 n.1, p.53-68.

[4] Miron Livny, Myron Melman, “Load balancing in homogeneous broadcast distributed systems”, *Proceedings of the Computer Network Performance Symposium*, p.47-55, April 13-14, 1982, College Park, Maryland, United States.
[5] H.S. Stone, “Critical Load Factors in Two-Processor Distributed Systems,” *IEEE Trans. Software Eng.*, vol. 4, no. 3.
[6] H.S. Stone. “Multiprocessor scheduling with the aid of network flow algorithms*”. IEEE Trans of Software Engineering*, SE-3(1):95--93.