

Evaluation of Load Balancing Strategies

Abubakar, Haroon Rashid and Usman Aftab

bakar@bakars.com, haroon@ciit.net.pk and awan@ciit.net.pk

COMSATS Institute of Information Technology, Abbottabad, Pakistan.

Abstract: *The distributed memory parallel processing technology encourages researchers to attack computationally intensive problems using inexpensive network of workstations. But unfortunately, distributed applications often face the problem of load imbalance in a heterogeneous environment. So, many algorithms / techniques were proposed [7 and 8] to manage / balance imbalanced load. And these algorithms differ from each other on the basis of certain controlling parameters.*

In this paper we have studied behaviors of a few such algorithms, i.e. fixed granularity, variable granularity (guided self scheduling) and global centralized task migration [4] algorithms. And the results yielded from this study demonstrate performance of the said algorithms, each in a certain condition.

This study was carried out with matrix multiplication as benchmark application on a NOWs using PVM as parallel library.

Keywords: *NOWs (Network of Workstations), PVM (Parallel Virtual Machine), Guided Self Scheduling, Fixed granularity, Variable Granularity, Granularity, Static Load balancing, Dynamic Load Balancing.*

1. INTRODUCTION

The users execute their processes on their hosts, and the random arrival of newly created processes can make some systems overloaded and some systems under loaded or idle. To overcome this problem at runtime, load balancing is used. Load Balancing is used in distributed memory multiprocessor to balance the workload among workstations automatically at execution time [6]. Ideally we always wish all processors to be executing continuously on the tasks to give a superior performance.

There are well known algorithms for load balancing that are mainly classified into two categories, static and dynamic load balancing. Considering the non homogeneous nature of the application and the runtime performance of the workers, dynamic load balancing shows better performance in most of the cases.

We compared three dynamic load sharing algorithms, i.e. fixed granularity, variable granularity and Global centralized load balancing [1]. The algorithms use their controlling parameters to increase the performance of parallel computation by carefully setting these parameters. In this paper, after studying the characteristics of these

parameters, we implemented them in certain scenarios and finally compared them in relevance to each other.

2. LOAD BALANCING

2.1 Static load balancing

This is the basic method of load balancing. In this method the performance of the workers is determined at the commencement of execution. Then depending upon their performance the work load is distributed in the start. The workers compute their assigned work and submit their result to the master.

The advantage of this method is less communication between the master and the workers which boosts the performance. The drawback is that it cannot adjust the runtime performance of the workers and non homogeneous nature of the application.

2.2 Dynamic load balancing

Dynamic load balancing determines the distribution of workload at run-time. The master assigns new task to the worker depending on the recent information collected. Since the workload distribution is done during runtime, it may give better performance. But the performance gained is at the cost of overhead associated with communication. So, the overhead associated should be in reasonable limit to achieve better performance [3].

There are many dynamic load balancing algorithms proposed. Some of the algorithms give better performance for short parallel jobs and others for large parallel jobs. But there are four basic steps that nearly all algorithms have in common [5]:

- 1) Monitoring workstation performance (load monitoring)
- 2) Exchanging this information among workstations (synchronization)
- 3) Calculating new distributions and making the work movement decision (rebalancing criteria)
- 4) Actual data movement (job migration)

3. LOAD SHARING ALGORITHMS

Load sharing algorithms are based on master-slave concept. The master creates the workers and then decide how much of the work load is to be distributed among the workers. The work load distribution to the workers is decided on the basis of a certain criteria.

Depending upon the criteria the load sharing algorithms are categorized into three types- Fixed granularity, Variable granularity, and Adaptive granularity. In fixed granularity approach, the task size is fixed before execution, and the same task size is assigned to each worker when it submits the result. In variable granularity the task size decreases continuously from beginning to the end of execution. Adaptive task granularity takes current system load into consideration for work load distribution [2].

3.1 Load Sharing Fixed Granularity Algorithm

In this algorithm, the task size is set and is assigned to the worker each time when the worker reports its result to the master. The task size is fixed through out the distribution of workload [2].

3.2 Load Sharing Variable Granularity Algorithm

Dandamudi describes about this algorithm in the following way [2].

“The difference between fixed and variable task granularity algorithms is that the task size decreases with time. For example, first task may have a size of 10 columns while the last one may have only one column. This can improve performance in two ways:

- 1) These algorithms attempt to minimize the time the master has to wait to receive the last results as task sizes at the tail end are much smaller.
- 2) It is ill-known that sending several short messages results in worse performance than when a small number of large messages is sent. In case of matrix multiplication, depending on the parameters used, variable task size algorithms can potentially decrease the number of messages sent while increasing their average size.”

There are many ways to decrease the task size. The one proposed by [6] is called Guided Self Scheduling algorithm. The task size in this algorithm is a function of remaining work. The function is generally $1/C$, where C is a constant.

3.3 Global Centralized Load Balancing Algorithm

Lee in his paper [4] proposed the algorithm which is global and centralized, in the sense that the load balancing decision is made by the master and the load is distributed according to the performance of the workers. The algorithm uses lockstep concept. This algorithm works in three phases e.g. computation phase, data distribution phase, and synchronization phase. In the first phase, all the workers will compute and there will be no communication. In data distribution phase, each task will distribute data to other tasks that requires it for the next lockstep. This phase confirms that all the nodes have reached a synchronization point.

[4] has introduced rebalancing criteria for distribution of data to take the correct decision and has introduced some parameters to explain the algorithm [1].

Tcompute_i: This is the interval between the start of execution of the first task and the time at which the last task completes for synchronization.

Task_i: It is average computation time for each task on the given workstation.

$$Ttask_i = Tcompute_i / n_i$$

Where n_i is number of tasks on the workstation i .

Thigh: The maximum $Tcompute$ over all workstations.

$$Thigh = \max\{Tcompute_i\} (1 \leq i \leq P)$$

Here P is the no of processors.

Tlow: The minimum of $Tcompute$ over all workstations.

$$Tlow = \min\{Tcompute_i\} (1 \leq i \leq P)$$

Here P is the no of processors.

For load balancing, tasks from workstations with longer $Tcompute$ should be moved to the workstation with shorter $Tcompute$.

Rebalancing Criteria: The rebalancing criterion is [1]:

$$Ttask_k \times n_k > \min\{Ttask_i \times (n_i + 1)\} (1 \leq i \leq P, i \neq k)$$

Here k is the index of the task with highest $Tcompute$, P is the no. of processors and n is the current no of tasks assigned to a workstation. After each lockstep, master checks this criteria and if it is true, then some tasks will be moved from the slow workstation to fast workstation.

4. RESULTS AND ANALYSIS

We wrote a simple matrix multiplication application, in C using PVM, for each of the three algorithms. The results were collected i.e. execution time for each algorithm for different values of the parameter. 400x400 matrices of type double were used through out the experiment.

Our experiments were carried out on 4 nodes, i.e. three dedicated workers and one master plus the worker. The machines that we used are

- 1) Be-Wulf (master + worker)
- 2) Ragamuphin (worker)
- 3) Basri (worker)
- 4) Lionking (worker)

4.1 Load Situation

As the machines in our lab were not very much loaded, so we had to create some background load in some machines.

“Be-Wulf” was being used as both, the master and the worker, so we did not create any load there. We created background load in other machines. For that we wrote an application that executes infinitely to keep the processor busy. Many instances of the same application were run on the three dedicated workers, to produce the background load.

4.2 Fixed Granularity Algorithm with Different Task Size

Task size is the workload distribution criteria in fixed granularity algorithm. The performance of the application varies depending on the task size. While, the no. of rows of the first matrix is the task size. Figure 1. shows the execution time for different task size.

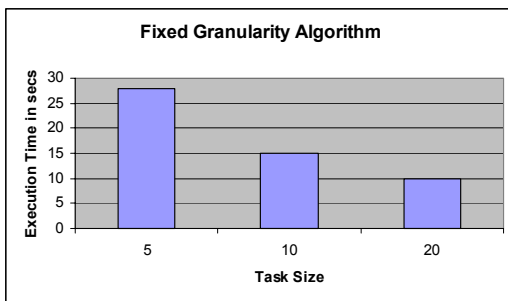


Figure 1. Performance variation of fixed.

We can see the execution time decreases as the task size increases from the Figure 1. It makes sense that larger task size needs less communication. But, if the task size is too big and slower machines become heavily loaded, that may degrade the performance of the system.

4.3 Variable Granularity Algorithm with Different Ratio

Guided Self Scheduling algorithm with different ratio for decreasing the task size is implemented and Figure 2. shows the result of the experiments.

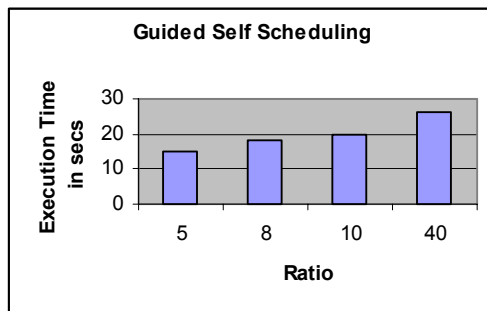


Figure 2. Guided Self Scheduling algorithm with different ratio.

The result shows that increasing task size increases performance. In Figure 2., increase in the ratio, which is decrease in the task size takes more execution time.

4.4 Global Centralized Load Balancing Algorithm with Different Lock Steps

The algorithm global centralized load balancing has two controlling parameters. One is the no of lock steps and the other is the no of tasks moving from one node to another after every lockstep. We fixed the second parameter as 4 i.e. after every lock step, 4 rows will be migrated from the slower node to faster node. The results were collected for different lock steps values.

Increase in the no of lock steps consumes more time. But, if the machines are more heterogeneous, then increase in the no. of lock steps may give better performance.

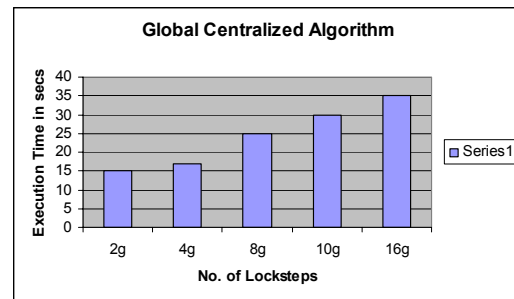


Figure 3. Global Centralized Load Balancing Algorithm with Different Lock Steps.

5. COMPARISON AMONG THREE ALGORITHMS

To reach concrete conclusion we need a perfect heterogeneous environment with huge load imbalance. So, as we do not have perfect heterogeneous environment with much load imbalance in our lab we compared the algorithms based on some assumptions.

For Fixed granularity we chose the task size as 5. This is chosen mainly because as there is no load rebalancing, so it may be inefficient to choose a large number of tasks.

For Variable granularity, we chose 10 as the ratio. Because, if the ratio is too big, then after two or three assignments, the task size will be the same. If the ratio is too small, the risk of assigning a very big task to one node which is slow is always there, which degrades the performance.

For global centralized load balancing algorithm proposed by [4], we chose the lock step as 4 on the assumption that too many lock steps may create overhead every time.

The comparison is shown in Figure 4., considering the above assumptions.

The graph shows that Variable granularity performs much better than fixed granularity and the global centralized load balancing algorithm proposed by [4] performs a little bit better than variable granularity.

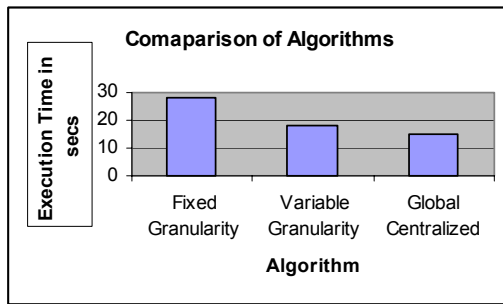


Figure 4. Comparison of algorithms.

6. CONCLUSION

The performance of fixed granularity algorithms increases with increase in task size.

The performance of Guided Self Scheduling algorithm increases with decreasing the ratio.

The performance of the algorithm proposed by [4] decreases with increasing no of lock steps.

The performances of all these three algorithms much depend on the parameters. But if the parameters are carefully chosen, Guided Self Scheduling algorithm and the algorithm proposed by [4] may provide better performance than fixed granularity algorithm.

REFERENCES

- [1] S. Malik, "Dynamic Load Balancing in a Network of Workstation", *95.515 Research Report*, 19 November, 2000.
- [2] A. Piotrowski and S. P. Dandamudi, "A comparative study of Load Sharing on Networks of Workstations", *Int. Conf. Parallel and Distributed Computing Systems*, pp.458-465, New Orleans, Louisiana, October 1997.
- [3] S.P. Dandamudi, "Sensitivity evaluation of dynamic load sharing in distributed systems", *IEEE Concurrency* 6 (3) (1998) 62-72.
- [4] Lee Bu Sung Francis, Cai Wen Tong, Heng Chee Khoo Alfred, "Dynamic load balancing in a message passing virtual parallel machine", *1st International Workshop on High-Speed Network Computing* (in conjunction with the 9th International Parallel Processing Symposium), 24 Apr 1995 - 28 Apr 1995, USA.
- [5] M. Zaki, W. Li, and S. Parthasarathy. "Customized dynamic load balancing for a network of workstations". *Journal of Parallel and Distributed Computing: Special Issue on Performance Evaluation, Scheduling, and Fault Tolerance*, June 1997.
- [6] C. Polychronopoulos and D. Kuck, "Guided self-scheduling: A Practical Scheduling Scheme for Parallel Computers", *IEEE Trans. Computers*, Vol. C-36, No. 12, pp. 1425-1439, December 1987.
- [7] G. Cybenko, "Dynamic load-balancing for distributed memory multicomputers", *Journal of Parallel and Distributed Computing*, (7)2, pages 279-301, October 1989.
- [8] Chengzhong Xu , Chenzhong Xu , Francis C. Lau, "Load Balancing in Parallel Computers: Theory and Practice", Kluwer Academic Publishers, Norwell, MA, 1997.