World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:5, 2007

Dynamic Load Balancing in PVM Using Intelligent Application

Kashif Bilal, Tassawar Iqbal, Asad Ali Safi and Nadeem Daudpota

***Abstract*---** This paper deals with dynamic load balancing using PVM. In distributed environment Load Balancing and Heterogeneity are very critical issues and needed to drill down in order to achieve the optimal results and efficiency. Various techniques are being used in order to distribute the load dynamically among different nodes and to deal with heterogeneity. These techniques are using different approaches where Process Migration is basic concept with different optimal flavors. But Process Migration is not an easy job, it impose lot of burden and processing effort in order to track each process in nodes. We will propose a dynamic load balancing technique in which application will intelligently balance the load among different nodes, resulting in efficient use of system and have no overheads of process migration. It would also provide a simple solution to problem of load balancing in heterogeneous environment.

***Keywords*—**PVM, Load Balancing, Task Allocation, Intelligent Application.

I. Introduction

PVM provide a platform for the distributed and parallel computing, and enables users to write parallel and distributed applications [1, 13]. Lot of work has been done already in PVM environment. Where two major things are focused regarding efficiency and throughput, these are Heterogeneity and Load Balancing**.** These are very important and critical areas needed to be focused while achieving the optimal results in distributed environment. [14] In load balancing different approaches were used where Process Migration is basic concept with different optimal flavors. Different techniques were adopted to accomplish task of load balancing e.g. *Central Algorithm* [2, 3,4], *The Rendez-Vous Algorithm* [5, 6], *The Random Algorithm* [7], *The Rake Algorithm* [6] etc**.**

Kashif Bilal is with the COMSATS Institute of Information Technology Abbottabad, NWFP, Pakistan. (e-mail: kashifbilal@ciit.net.pk & phone: 0092 300 5613174).

Tassawar Iqbal is with the COMSATS Institute of Information Technology Abbottabad, NWFP, Pakistan. (e-mail: tassawar@ciit.net.pk & phone: 0092 300 5137071).

Asad Ali Safi is with the COMSATS Institute of Information Technology Abbottabad, NWFP, Pakistan. (e-mail: safi@ciit.net.pk & phone: 0092 300 5912660).

Nadeem Daudpota is with the COMSATS Institute of Information Technology Abbottabad, NWFP, Pakistan. (e-mail: daudpota@ciit.net.pk & phone: 0092 992 383591).

Most of these existing load balancing techniques use process migration between nodes. But Process Migration is not an easy task; it imposes a lot of burden and processing effort in order to track each process in nodes. Mostly check point-restart mechanism is used [8, 9], where check pointing of a process basically boils down to writing the address space of a process to a file and then receiving its contents afterwards (mapping it to memory). This includes the shared libraries, [15] which may be used by the process. In addition, the contents of (some of the) processor registers have to be taken care of, such as the program-counter and the stack pointer. [10]. If a running task is migrated, special care has to be taken regarding messages routed towards that task, i.e. Routing information should also be updated in nodes [8]. Most of the dynamic loads balancing techniques are restricted for homogeneous systems. Processes running in a heterogeneous environment can not be migrated with ease, and in, most cases processor architecture and operating system should be same [10]. Also processes using external resources e.g. sockets, are difficult to migrate even in homogenous environment and needs special techniques handle external resource references at receiver node e.g. using a virtual operating system [11]. Our proposed strategy is to make intelligent decision before spawning the slave processes in a Master Slave model, so that process of task assignment becomes automatically balanced and there would be no need for process migration and handling heterogeneity problems.

II. Intelligent Load Balancer

Our proposed load balancing scheme has intelligence at application level. This scheme can be divided into 3 major parts,

i) Getting information about processor's current load status and processor power from slave nodes and sending that information to Master node.

ii) Collecting information sent from slave node and making intelligent decision based on collected information for task assignment for each node.

iii) Spawning new tasks at nodes, based on decision made by scheduler algorithm.

A Status Agent

A simple daemon program would be running at each node responsible for providing current status of processor and power of processor, and sending it to master node, so that scheduler running at master node can take decision about load assignment to nodes. Different techniques can be used to collect information form local processor, e.g. we can use *vmstat* command. There are numerous approaches for determining the load of a computer under a valid login without root privileges. For simplicity and portability, we chose the *vmstat* command which provides the percentage of CPU time for all processes decomposed into user, system, and idle time. The user time is the percentage of CPU utilization observed at the application level during a specified time interval. The system time is the percentage of CPU utilization by kernel during a specified time interval. The idle time is percentage of time CPU is observed to be idle during a specified time interval. [12].

B Intelligent Scheduler

A scheduler would be running at master node. In PVM we have a Master process that is initiated manually at some node. When this process would be initiated, it would spawn status agent processes at slave nodes to collect the local information regarding each node. These status agents will send their information to scheduler running on master node and would be terminated, because we don't need them after provision of local node statistic. After receiving the status from each slave node, now scheduler will decide that how much workload should be assigned to individual node.

This calculation of load distribution is critical and varies from homogeneous (same Processing Power) to heterogeneous (different processing power). Intelligent scheduler performs these mathematical calculations as follows;

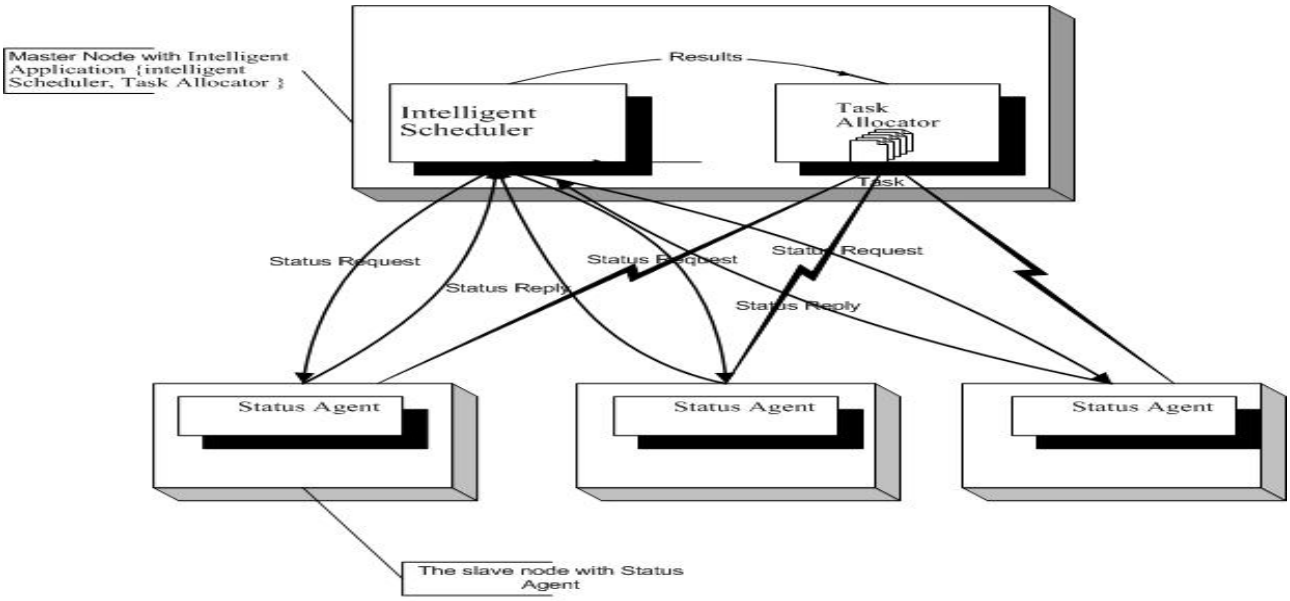


Figure 1: The block diagram of the intelligent application is shown, where intelligent application is decomposed into three major components. Two major components of application reside in the Master node. Those are Intelligent Scheduler and Task Allocator. Third component reside in every node called Status Agent. The Intelligent Scheduler initiate task on each node to get the current status of the each node that is accomplished through the Status Agent. Status Agent in turn returns the results to the Intelligent Scheduler that decide the distribution of work load on node. Task allocation is done finally through the Task Allocator.

1) Mathematical Calculations for Homogeneous Environment

In this environment the intelligent Scheduler, collects the idleness of each processor at node through the Status Agent and mathematically calculates the relative idleness of each node. On basis of these calculations assign work load to each node as per its capabilities.

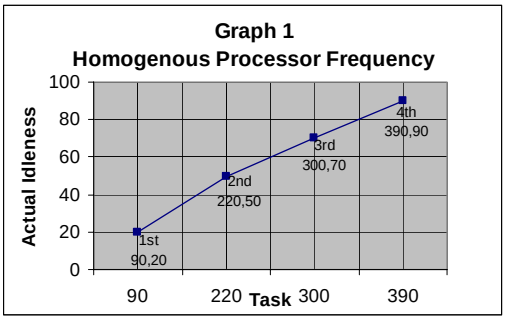
Let we have four nodes in the environment each having same processor frequency. Status Agent on request returns the idleness of each node, e.g. 1^{st} node has 20% idleness, 2^{nd} has 70% idleness, 3^{rd} has 50% idleness and 4^{th} has 90% idleness. After these statistics the Intelligent Scheduler calculates the relative idleness of each node.

Total Idleness = $1^{st} + 2^{nd} + 3^{rd} + 4^{th}$

230 = 20 + 70 + 50 + 90

Relative Idleness of 1st node would be = (20 / 230) * 100 = 8.69 (approx 9%).
Relative Idleness of 2nd node would be = (70 / 230) * 100 = 30.43 (approx 30 %).
Relative Idleness of 3rd node would be = (50 / 230) * 100 = 21.73 (approx 22%).
Relative Idleness of 4th node would be = (90 / 230) * 100 = 39.13 (approx 39%).

These results will be handed over to the Task Allocator. Let 1000 records are available to be processed. On the basis of these results, the Task Allocator will assign the 90 records to 1st node for processing, 300 records to 2nd node for processing, 220 records to 3rd node for processing and 390 records to 4th node for processing. This assignment is shown in Graph 1.
On the basis of these mathematical results our application intelligently distributed the work load among the nodes. As a consequence now all the nodes will finish their assigned task almost at same time. This is the actual beauty of our application. Where no need to migrate the task once they are executed to balance the load as load would be already balanced.



2) Mathematical Calculations for Heterogeneous Environment

In this environment where the each node has different Processor Frequency but all nodes have same idleness, here the Intelligent Scheduler will collect the Frequency of each processor at node through the Status Agent and mathematically calculates the relative Frequency of each node. On basis of these calculations assign work load to each node as per its capabilities.
Assume that we have four nodes in the environment each node has same idleness in percentage. Status Agent on request returns the Processor Frequency of each node. Let 1st node has 500MHz, 2nd has 400MHz, 3rd has 1000MHz and 4th has 900MHz. After these statistics the Intelligent Scheduler calculates the relative Processing Power of each node.

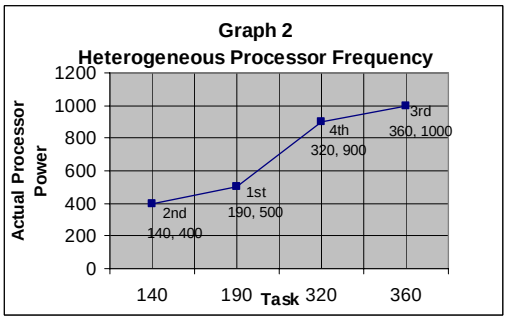
Total Power = 1st + 2nd + 3rd + 4th

2800 = 500 + 400 + 1000 + 900

Relative Power of 1st node would be = (500 / 2800) * 100 = 17.85 (approx 18%).
Relative Power of 2nd node would be = (400 / 2800) * 100 = 14.28 (approx 14%).
Relative Power of 3rd node would be = (1000 / 2800) * 100 = 35.71 (approx 36%).
Relative Power of 4th node would be = (900 / 2800) * 100 = 32.14 (approx 32%).

Likewise these results will be handed over to the Task Allocator. Let 1000 records are available to be processed. On the basis of these results, the Task Allocator will assign the 180 records to 1st node for processing, 140 records to 2nd node for processing, 360 records to 3rd node for processing and 320 records to 4th node for processing. This assignment is shown in Graph 2.

Using these mathematical results our application intelligently distributed the work load among the nodes. As a consequence now all the nodes will finish their assigned task almost at same time.



C Task Allocator

Now tasks would be spawned at each slave node based on decision made by scheduler and that would be balance automatically.

III. Intelligent Application Operation

In order to explain the working of proposed idea, let we have image recognition example. In this scenario we are asked to match a particular thumb expression with the records being stored in the database in order to identify the authentication of a person. For the simplicity we assume that database is populated with the 10000 records. First of all the Intelligent Scheduler at Master Node would spawn status agent processes at slave nodes to collect the local information regarding each node. Status Agent at each node collects the necessary status of that node that is Idleness in percentage. However the status agent not only take the value at single instant of time using vmstat command but it take status information for ten instances of time (ten seconds). Then it takes the average of

these ten status values and returns a single value to Intelligent Scheduler. Reason to take the multiple status values is, to precise the results. Every Status Agent performs this operation on each node and hand over the results to the Intelligent Scheduler. On the basis of these results the scheduler makes the decision about the load distribution so that each node is given the task as per node's capabilities. This decision making process is very critical. Intelligent Scheduler would perform this through an algorithm. That is a bit complex in nature. If we have the each node with the same idleness or if we have the same processing power, its decision could be done very simply but this never happens practically because each node has in practical environment has different idleness status as per its workload and also posses the different processing powers, e.g.; one node has processor of 1000MHz with idleness of 70%, other has 500MHz with idleness of 90% and so forth. In this decision making becomes very critical and difficult. Keeping in mind all these demands, algorithm makes the distribution of the task and sends the results to the Task Allocator. On the basis of these results the Task Allocator performs the task distribution. In this example we have 10000 records and we assume that we have 10 nodes in environment. The Task Allocator divides the data in chunks as per load capabilities of the each node, and assigns the load to the respective nodes, whole mechanism is shown in Figure 1.

IV. Conclusion

Concluding, the concept of load balancing is achieved without migration of the processes among the node instead intelligent task assignment is being done automatically in order to balance the workload and there would be no need for process migration and handling heterogeneity problems. In past approaches of load balancing, there were some drawbacks and shortcomings. Few of them are discussed; Most of time waste in check-point restart mechanism and in migration of processes among the nodes. As our proposed system performing load balancing without the migration hence the problems of migrating a process in heterogeneous environment heterogeneity are removed automatically. Also if any external resource is required like socket there is no need to deal with them as there would be no process migration. It make possible to achieve the load balancing without having the need to maintain tables for the process migration at each node as a result lot of processing efforts are saved.

References

[1] V. S. Sunderam, "PVM A framework for parallel distributed computing", Concurrency, Practice and Experience by John Wiley & Sons Vol. 2(4), pages 315--339, December, 1990.
[2] A. Osman, H. Ammar, "Dynamic Load Balancing Strategies for Parallel Computers", International Symposium on Parallel and Distributed Computing (ISPDC), Romania, July 2002.
[3] Hillis, W.D. "The Connection Machine" MIT press, Cambridge, 1985.
[4] Powley, C., Ferguson, C. and Korf, R. E. "Depth-First Heuristic Search on a SIMD Machine", Artificial Intelligence, vol. 60, pages.199-242, 1993.
[5] Fonlupt, C., Marquet, P. and Dekeyser, J. "Data-parallel load-balancing strategies", Parallel Computing, pages1665-1684, 1998.
[6] Dekeyser, J. L., Fonlupt, C. and Marquet, P. "Analysis of Synchronous Dynamic Load Balancing algorithms, Parallel Computing", State-of-the Art Perspective (ParCo'95), vol. 11 of Advances in Parallel Computing, pages 455--462, Gent, Belgium, September 1995.
[7] Subramanian, R. and Scherson, "An Analysis of Diffusive Load Balancing". Proceedings of Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, pages 220—225, June 1994.
[8] Leen Dikken, Frank van Der Linden, Joep Vesseur and Peter Sloot, "Dynamic PVM", Parallel Scientific Computing and Simulation Group, Springer Verlag, April 1994.
[9] M.J Litzkow, M.Livny. "Condor – A hunter of idle workstation", 8th International Conference on Distributed Computing Systems, San Jose, California, June 1988.
[10] K.A. Iskra, Z.W. Hendrikse, G. D. van Albada, B.J. Overeinder, P.M.A Sloot, "Experiments with Migration of PVM Tasks", Proceedings of the sixth annual conference of the Advanced school for Computing and Imaging ASCI , June, 2000.
[11] Ravikanth Nasika Partha Dasgupta "Transparent Migration of Distributed Communicating Processes", 13th ISCA International Conference of Parallel and Distributed Computing Systems, Aug, 2000.
[12] David J. Jackson Chris W. Humphres, "A Simple Yet Effective Load Balancing Extension to the PVM Software System", Parallel Computing, Volume 22, pages 1647-1660, February, 1997.
[13] A.I Geis, "Building a Foundation for the Next PVM", Petascale Virtual Machines, Springer-Verlag, London, 2001
[14] Dennis Guster, Abdullah Al-Hamamah, Paul Safonov, Elizabeth Bachman, "Computing and network Performance of a distributed parallel processing environment using MPI and PVM communication methods", Journal of Computing Sciences in Colleges USA April 2003
[15] Arthur Trey, Nelson Michael L. "Intel NX to PVM3.2 Message Passing Conversion Library", NASA Langley Technical Report Server, October 2003.