Profile-Based Load Balancing for Heterogeneous Clusters *

M. Banikazemi, S. Prabhu, J. Sampathkumar,
D. K. Panda, T. W. Page and P. Sadayappan

Dept. of Computer and Information Science
The Ohio State University
2015 Neil Ave.
Columbus, OH 43210

Contact Author: M. Banikazemi (banikaze@cis.ohio-state.edu)

1. Introduction

Cluster computing is becoming increasingly popular for providing cost-effective and affordable parallel computing for day-to-day computational needs [2, 11, 16]. Such environments consist of clusters of workstations connected by Local Area Networks (LANs). The possibility of the incremental expansion of clusters by incorporating new generations of computing nodes and networking technologies is another factor contributing to the popularity of cluster computing. However, this incremental expansion leads to heterogeneity in speed and communication capability of workstations, systems with different memory and cache organization, coexistence of multiple network architectures, and availability of alternative communication protocols. This is forcing the cluster computing environments to be redefined as heterogeneous clusters.

Most of the current research on cluster computing is directed at developing better switching technologies [1, 6, 17] and new networking protocols [9, 10, 18] in the context of homogeneous systems. The effect of heterogeneity in communication has been studied recently to develop efficient collective communication algorithms [3] for heterogeneous clusters. Heterogeneity also has an effect on load balancing schemes making them even more difficult. Efficient scheduling of applications based on the characteristics of computing nodes and different segments of applications have been studied for heterogeneous system [8, 13, 15]. An application-level scheduling scheme has also been proposed [4, 5]. One of the common assumptions involved in all these works is that the execution times of various fractions of an application scale up linearly with the fraction size. However, this is not a realistic assumption for many applications on a given workstation. **Execution times for different applications exhibit nonlinearity with respect to problem size and different fractions of the problem.** The nonlinearity also varies across workstations. This leads to a challenge whether efficient and optimal load balancing schemes can be developed for heterogeneous clusters by taking these nonlinearities into account.

In this paper we take on this challenge. We first characterize the performance of the computing nodes with respect to the size of the problem assigned to them. Then, we propose a new algorithm to distribute the work among computing nodes more accurately and minimize the overall execution time. We show

how the performance of different machines can be extrapolated with respect to the problem size under the absence or the presence of external load. We consider different application programs and workstations to evaluate the proposed scheme. We show that by adopting the proposed framework, reductions up to 46% can be obtained in overall execution time.

2. Nonlinearity in Execution Time

In order to distribute a problem (in a balanced manner) among a set of nodes, we need to know the execution times for various fractions of the problem on the participating nodes. For predicting these execution times, most of the current load balancing schemes (both static and dynamic), rely on using the execution time of a certain fraction of the problem. They estimate the execution times of other fractions by implicitly assuming that the execution time is a linear function of the fraction of problem size. But now the question is whether this assumption holds true for different applications and different platforms.

Figures 1a and 1b show the execution times of two different applications on different platforms. The first observation that can be made is that the performance of the computing nodes depends on the application being executed. This leads us to conclude that the behavior of computing nodes cannot be characterized by a single number independent of the application. In other words, measures such as the CPU speed are not accurate indicators of the performance of nodes in a cluster. Secondly, the execution time as a function of problem size is not linear. Furthermore, even for a given problem size the execution time for computing a fraction of the problem is not a linear function of the fraction (Fig. 1c). Hence, for a given problem size, estimating the execution times of various fractions based on that of a certain fraction may result in inaccuracies. Thus, we need more sophisticated methods capable of predicting execution times precisely.

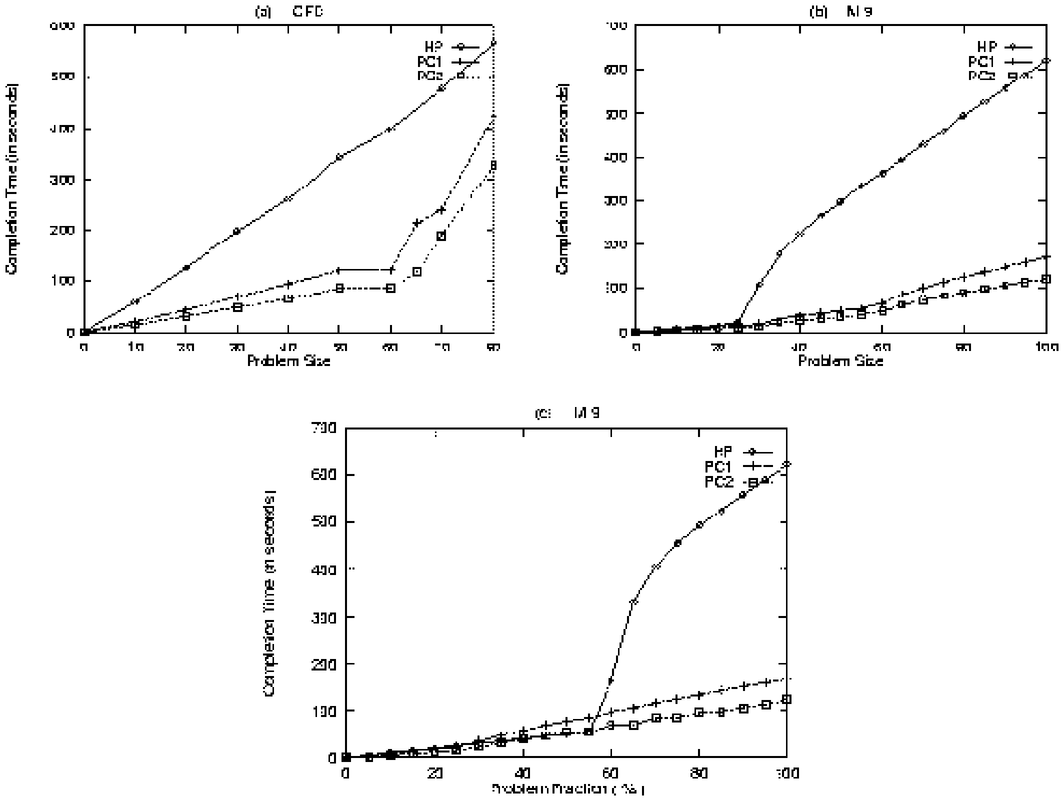

Figure 1: (a) and (b) represent the execution times of two programs for different problem sizes. CFD is a Computational Fluid Dynamics application, and M9 calculates M^9 for square matrix M of size up to 400x400. (c) represents the execution times of different fractions of M9 problem for a matrix of size 400x400.

There has been some earlier work in which more than one factor (i.e., processor speed and memory size) have been used in order to make predictions about execution times [20]. However, to the best of our knowledge, no systematic method has been proposed for accurately characterizing the effect of all issues on the execution time of a given problem. In the next section we propose a new method for characterizing the behavior of a node with respect to an application program and use this characterization to develop a (nearly) optimal load balancing scheme.

3. A Profile-Based Approach to Load Balancing (PBLB)

In the previous section we argued that the performance of a computing node can not be accurately characterized by a single number (i.e., a linear curve). In order to predict the execution time of an application more accurately, more information is needed. Obviously, it is not possible to obtain the execution times of all possible data sizes of an application in order to have the desired optimal load balancing. However, we show that having more data points (i.e., having the execution times for different fractions of the problem) increases the accuracy of load balancing schemes and can lead to significant improvements.

We first propose a new algorithm for finding the (near) optimal solution for balancing the load between two nodes. Then, we show how this algorithm can be applied to systems with an arbitrary number of nodes. We assume an SPMD model of computation. Furthermore, we assume that the application can be broken into smaller units of computation (parallelism comes from do-all loops). Therefore, we essentially consider the problem of efficiently mapping the data partitions among a set of heterogeneous computing nodes, so as to minimize the execution time of the application.

3.1 Basic Approach

Observation 1: In the optimal case, and where arbitrarily small or large loads can be assigned to a computing node, the execution times of different portions of an application program on all participating computing nodes are identical.

Now, we present the PBLB algorithm, which given S sample points (execution times for S different fractions of a problem) for different computing nodes in a cluster, can find the best distribution of the work among these nodes. For now, we assume that no communication between nodes is required. Furthermore, we start with a simple case in which there are only two nodes in the system.

Let us assume that we have the execution times of different fractions of an application for the computing nodes: A and B (Table 1). We start by considering dividing the load evenly between these two nodes. By referring to the table, we know Nodes A and B need 45 and 80 seconds for executing 50% of the load, respectively. From Observation 1, we know that for finding the optimal solution, we need to assign more work to node A and less work to node B. In other words, the share of load for nodes A and B should be in the 50%-100% and 0%-50% range, respectively. As a next step, we consider the execution times of the fractions in the middle of these two ranges. The execution time for 70% of load on node A is 70 seconds, while that for the remaining (30%) on node B is 40 seconds. Since the execution time for node A is greater than that for node B, from Observation 1 we conclude that the optimal solution will be in 50%-70% and 30%-50% range on nodes A and B, respectively. Again, we consider the execution time corresponding to the middle points of these ranges: 55 and 65 seconds and conclude that an optimal distribution should be such that the shares of load for node A and node B reside in the 60%-70% and 30%-40% ranges, respectively. We assume that the execution time vs load percentage curves are piece-wise linear. Using this assumption, we find the estimated optimal execution time to be 58.75 seconds corresponding to 62.5% load on node A and 37.5% load on node B. It should be noted that in practice, we may need to round off the calculated percentages of load and therefore, the actual execution time may slightly differ from the estimated execution time.

The common approach in current static load balancing schemes is to consider the overall execution time of the problem and based on that derive the execution time of a particular fraction assuming that the execution time is a linear function of the problem fraction [20]. We refer to this approach as the "Standard" algorithm. It can be easily shown that if, for example, we use only the execution times of 100% load for finding the distribution, we will get execution time of approximately 70 seconds. Thus, using the proposed algorithm leads to 17% improvement in comparison with the standard algorithm. It should be noted that the complexity of this algorithm is $O(S \log_2 S)$, S being the number of samples.

Percentage of Load	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Node A	0	3	8	15	30	45	55	70	80	85	90
Node B	0	10	30	40	65	80	100	130	160	180	205

Table 1: Execution times of different percentages of an application on two different nodes.

3.2 Generalized Approach

Now, we extend the algorithm for cases where there are more than two nodes in the system. In most heterogeneous environments there are some identical computers in the system. We call each set of these identical nodes a class of nodes. Now, suppose that our cluster consists of C classes, and there are C_i nodes in class i. Since the nodes in a given class are identical, their behavior with respect to a given application program will be identical too. Thus, any class of nodes can be represented with one execution time table. Since, all the nodes in a class are identical, from Observation 1 we know that they have to be performing the same amount of work. Therefore, for a given application and problem size, the execution time of x percent of the problem on N machines of the same class, is equal to the execution time of (x/N) percent of the problem on a single class member. This time can be easily obtained from the execution time table of one of the class members.

Furthermore, we can combine two classes into one super-class. The execution time table of a super-class can be constructed by finding the best distribution of work between two classes for all samples (different percentages of the whole problem) by using the basic approach explained before. We can repeatedly combine classes until we finally get two super-classes (each with their own execution time table). Then, we can recursively descend the hierarchy allocating loads to classes using the basic approach. Finally, we obtain the share of each node by dividing the work assigned to a class equally among the nodes of that class. This scheme is illustrated in Figure 2.

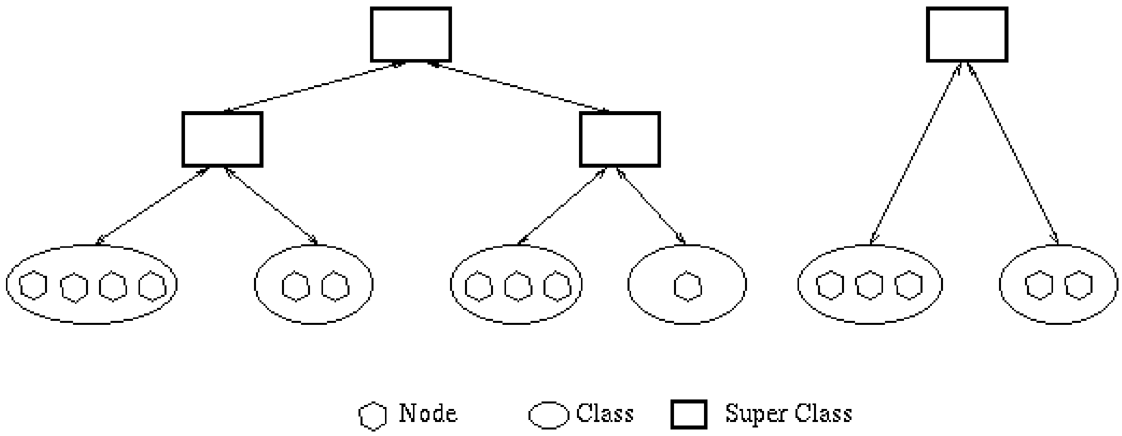

Figure 2: Combining different classes into super-classes.

The complexity of the PBLB algorithm is $O(C\ S\ \log_2 S)$, where C is the number of classes, and S is the number of sample points in the execution time table. This algorithm was tested for a large number of classes and the execution time of the algorithm was found to be of the order of a few milliseconds. It should be noted that if there is only one sample point in the execution time vs fraction size curve, our algorithm degenerates to those currently being used in static and dynamic load balancing schemes. In the following section, we show how our framework can be applied to systems with external load.

4. Effect of External Load

The existence of different external load on the nodes can lead to heterogeneity even in a homogeneous system. Therefore, in this section we extend PBLB to take into account the existence of external load while performing load balancing. Since the execution times for different samples could have been generated on systems without any external load (or with constant external load), it is important to have a mechanism to predict the execution time of an application on a machine with arbitrary amount of external load. If there are two processes running on a machine, the CPU share of each node is practically half. In other words, the execution time of each process is doubled in comparison to the case of a single process running. Figure 3 supports the above observation. The PBLB algorithm incorporates the effect of external load by calculating the scale up/down of the execution time of a certain fraction and then uniformly applying the same to all samples.

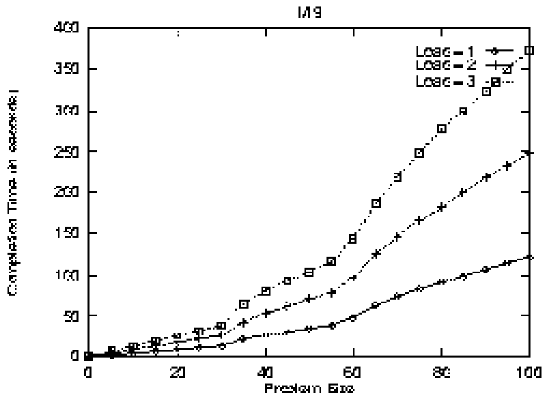

Figure 3: Effect of load on execution time: The three curves correspond to the execution time of the M9 application with no background process, 1 background process, and 2 background processes.

5. Experimental Evaluation

In order to evaluate the performance of the proposed algorithm, we implemented PBLB and the standard algorithm. We used these algorithms to perform load balancing of the M^n application on a heterogenous cluster of workstations. M^n is an essential part of the algorithm for computing all-pairs shortest paths [7]. We wrote the program as an SPMD application using the MPICH [12] implementation of the MPI [14] standard. The results were obtained for a matrix (M) size of 400x400 and n=9. Our testbed comprised of 12 nodes connected through switched Fast Ethernet. Four of the nodes were Pentium Pro 200MHz PCs with 128MB memory and 16KB/256KB L1/L2 cache. These machines are referred to as "slow nodes" in the rest of this section. The other eight nodes were Pentium II 300MHz PCs with 128MB memory and 32KB/256KB L1/L2 cache. We refer to these machines as "fast nodes". All these machines were running Linux 2.0.30.

5.1 Evaluation on Unloaded Machines

We compared the execution time of the PBLB algorithm and that of the standard algorithm. The total number of nodes and the relative number of fast and slow nodes were varied for the comparison. The normalized overall execution times are shown in Figure 4a. The three groups of bars correspond to total number of nodes being 4, 6 and 8. The x-axis shows the number of fast and slow (fast,slow) nodes. These experiments were run on idle machines with no other background processes. The times shown have been normalized with respect to the largest execution time in that particular group (same number of total nodes). The PBLB algorithm consistently performs better for all cases compared to the Standard algorithm. The percentage of improvement in comparison to the standard algorithm varies from 10-28%.

5.2 Evaluation on Loaded Machines

In section 4, we presented the effect of external load on the performance of the M^n program on a given machine. We measured the effect of background load on the relative performance of the PBLB and the standard algorithm on a system comprising of one fast and three slow nodes ((1,3) configuration as shown in Figure 4a). These results are shown in Figure 4b. The three groups of bars on this graph correspond to changing the external load on 1) only fast nodes, 2) both, and 3) only slow nodes. The groups of bars on the graph have been normalized with respect to the largest execution time in the group. The observed trend suggests that the more heterogeneous a system becomes, the more improvement we gain by using PBLB.

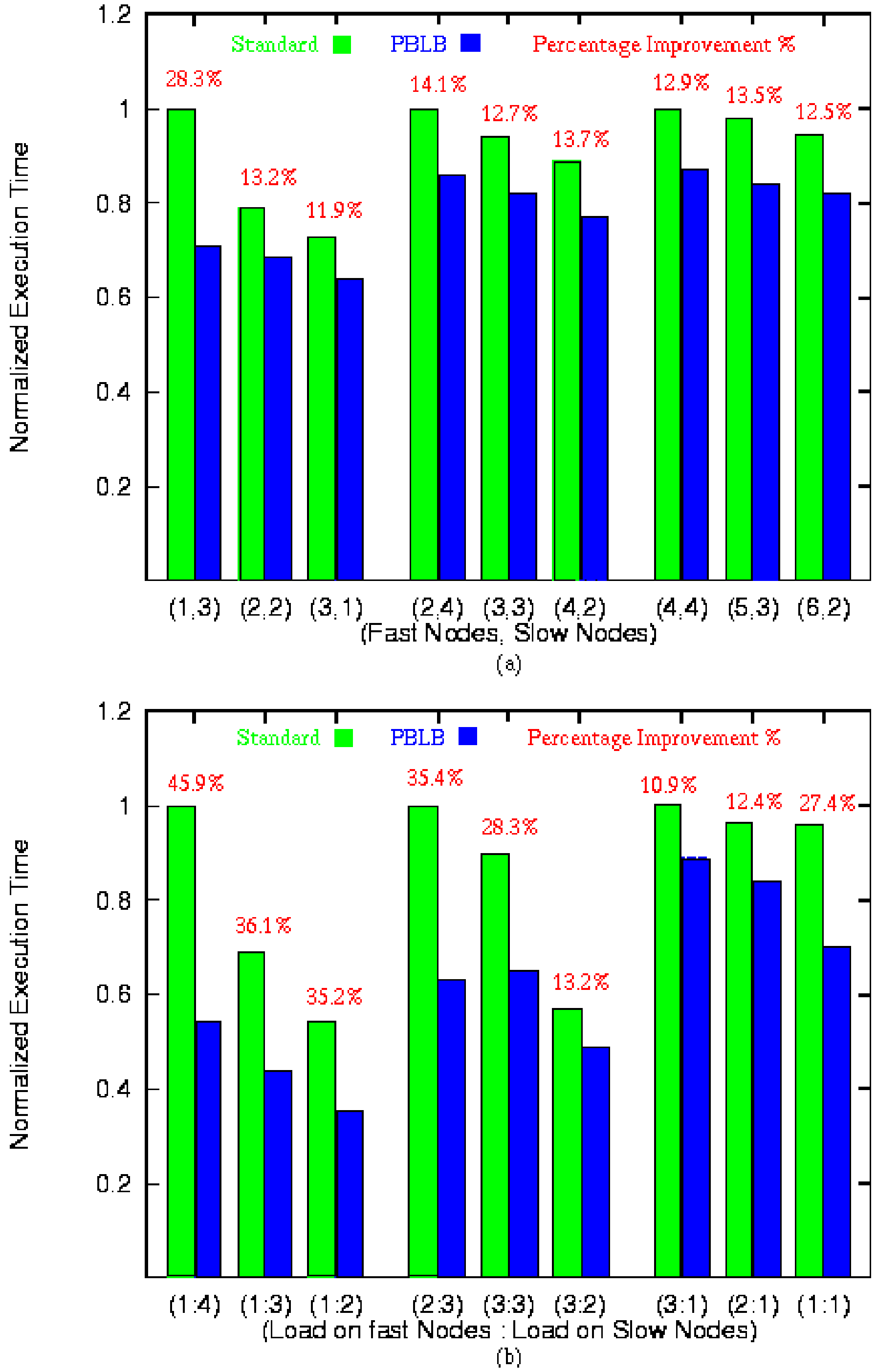

Figure 4: (a) Performance comparison of PBLB & Standard algorithms for a system of unloaded nodes. The 3 groups correspond to a system of 4, 6 & 8 nodes. (b) Performance comparison of PBLB & Standard algorithms for a system of 1 "fast" and 3 "slow" nodes with varying external load on the nodes. The percentage of improvement for each case is shown on top of the bars.

7. Conclusions and Future Research

In this paper, we have presented a new profile-based approach to perform load balancing in heterogeneous clusters. We have discussed the nonlinearity issues in execution times of different problems and problem sizes. We have proposed a new PBLB algorithm which uses application specific information (profile) to come up with a (near) optimal load distribution. We have incorporated the effect of external load into PBLB and hence come up with a complete strategy for static load balancing. Performance evaluation of this algorithm on a 12-node testbed indicates that with increased degree of heterogeneity, the execution times can be reduced by up to 46%.

In this paper, we have assumed the applications to be computation intensive and ignored the effect of communication costs. We are working towards extending PBLB to incorporate the effect of communication on load balancing. We are also looking at extending this framework to perform dynamic load balancing by using PBLB in the "Work Transfer Vector Calculation" phase [19] of the existing schemes. We will be presenting these results in the final version of this paper.

Footnotes

* This research is supported in part by NSF Grants CCR-9704512, IRI-9501812, and CDA-9514898 and the OCARNet grant from the Ohio Board of Regents.

References

[1] ATM User-Network Interface Specification, Version 3.1. ATM Forum, 1994.

[2] T. Anderson, D. Culler and Dave Patterson. A Case for Networks of Workstations (NOW). IEEE Micro, 95.

[3] M. Banikazemi, V. Moorthy and D. K. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. International Conference on Parallel Processing, 1997, accepted for presentation.

[4] F. Berman and R. Wolski. Scheduling from the Perspective of the Application. HPDC, 96.

[5] F. Berman, R. Wolski, S. Figueira, J. Schopf and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. Supercomputing, 96.

[6] N. J. Boden, D. Cohen et al. Myrinet: A Gigabit-per-Second Local Area Network. IEEE Micro, 1995.

[7] T. H. Cormen, C. E. Leiserson and R. L. Rivest. Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company, 1990.

[8] J. C. DeSouza-Batista, M. M. Eshaghian, A. C. Parker, S. Prakash and Y. C. Wu. A sub-optimal assignment of application tasks onto heterogeneous systems. In Proceedings of the Heterogeneous Computing Workshop, 1994.

[9] T. von Eicken, A. Basu, V. Buch and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. ACM Symposium on Operating Systems Principles, 1995.

[10] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. International Symposium on Computer Architecture, 1992.

[11] E. W. Felten, R. A. Alpert, A. Bilas, M. A. Blumrich, D. W. Clark, S. N. Damianakis, C. Dubnicki, L. Iftode and K. Li. Early Experience with Message-Passing on the SHRIMP Multicomputer. International Symposium on Computer Architecture (ISCA), 1996.

[12] W. Gropp, E. Lusk, N. Doss and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Argonne National Laboratory and Mississippi State University.

[13] C. Leangsuksun and J. Potter. Design and Experiments on heterogeneous mapping heuristiSC. In Proceedings of the Heterogeneous Computing Workshop, 1994.

[14] MPI: A Message-Passing Interface Standard. Message Passing Interface Forum, 1994

[15] B. Narahari, A. Yousef, and H.-A. Choi. Matching and scheduling in a generalized optimal selection theory. In Proceedings of the Heterogeneous Computing Workshop, 1994.

[16] V. S. Sunderam. PVM: A Framework for Parallel and Distributed Computing. Concurrency: Practice and Experience, 1990.

[17] C. B. Stunkel and D. G. Shea and B. Abali et al. The SP2 High-Performance Switch, IBM System Journal, 1995.

[18] S. Pakin, M. Lauria and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). Proceedings of the Supercomputing, 1995.

[19] J. Watts. A practical approach to dynamic load balancing. MS. Thesis, California Institute of technology, 1995.

[20] M. J. Zaki, W. Li and M. Cierniak. Performance Impact of Processor and Memory Heterogeneity in a Network of Machines. In Proceedings of the Heterogeneous Computing Workshop, 1995.