Communication Pattern Based Node Selection for Shared Networks

Srikanth Goteti
ComStock, Division of Interactive Data Corp
600 Mamaroneck Avenue
Harrison, NY, 10528
srikanth.goteti@cmstk.com

Jaspal Subhlok
University of Houston
Department of Computer Science
Houston, TX 77204
jaspal@uh.edu

Abstract

Selection of the most suitable nodes on a network to execute a parallel application requires matching the network status to the application requirements. We propose and validate a novel two step approach that exploits the knowledge of the communication structure of the application to address this problem. In the first step, a small set of candidate node groups are selected as potential sites of application execution, by analyzing the network status information and the communication patterns used by the application. The second step is based on the concept of a communication skeleton, which is a short running program that generates the dominant communication operations of the application it represents. The communication skeleton is executed on all candidate groups of nodes. The node group selected for application execution is the one that achieves the best performance on the communication skeleton. This approach leads to customized node selection and is particularly well suited to situations where available network information is of poor quality or expected communication performance cannot be modeled accurately. We motivate this approach, describe a prototype implementation, and present performance results for NAS Parallel Benchmarks executing on a shared network testbed.

1. Introduction

Selection of computation nodes to execute a parallel application is a central problem for computing on shared clusters and computation grids [7, 8]. Node selection based on CPU considerations has been addressed by several systems, some well know examples being Condor [9] and LSF [20]. The problem of node selection is significantly more complex when the communication needs of applications must also be taken into account. The main reasons for the additional complexity are that the communication properties cannot be associated with individual nodes, network status changes dynamically, and the availability of network resources is difficult to measure and predict accurately. The state of the art in resource selection with communication considerations can be paraphrased as follows. The status of the network is measured and predicted with tools such as Network Weather Service [19] and Remos [10, 11], and this information is analyzed to identify a good group of available computation nodes and network paths for application execution. Some research projects have focused on getting the best general group of execution nodes [3, 14, 18] while others have developed procedures customized for a particular application or application class [4, 5, 6, 12].

This approach to node selection has the fundamental drawback that the decisions made are, at best, only as good as the accuracy with which the network status was measured and future network performance predicted. The performance that a network delivers to an application can vary significantly from the performance predicted by network measurement tools for a variety of reasons, some of which are as follows:

- Network status and prediction information may be outdated. Measurement of network properties, such as available bandwidth, can be intrusive and expensive, and the cost rises rapidly with the size of the network. Hence, it may be practical only to perform measurements relatively infrequently while the network state changes continuously.
- Network tools typically measure the unused network capacity or the network bandwidth achieved by a specific measurement probe. However, the relationship between available network capacity and the performance achieved by communication operations in an application is complex, and depends on other factors also, such as the network transport protocols in use. For example, the bandwidth that a TCP stream achieves on a busy network route partly depends on the number of other TCP streams using the links in the route.

- The performance of collective communication operations, common in parallel applications, is very difficult to estimate on a shared cluster or a grid environment. We are not aware of any tools that have proven their effectiveness in this respect. In particular, interference between multiple application communication streams sharing the same network path, is very difficult to model.

The point is that the expected performance of an application's communication operations inferred from network measurement tools can be significantly different from the actual performance for several different reasons. This limits the effectiveness of any node selection procedure entirely based on network measurements.

This research pursues a new approach to node selection motivated by the above discussion. The centerpiece of our methodology is the concept of a *performance skeleton* of an application, which is defined as a synthetically generated short running program that has the same fundamental execution characteristics as the application it represents, but with no semantic relevance. The execution time of the performance skeleton program on a given set of nodes reflects the execution time of the application under the same conditions, but possibly scaled down by multiple orders of magnitude. In this approach, the performance of the performance skeleton on a group of nodes determines the likelihood of those nodes being chosen for execution of the corresponding application since the performance of the skeleton is closely related to the performance of the application. This methodology eliminates the impact of inherent inaccuracy in network measurement and modeling. A performance skeleton is constructed ahead of time and executed prior to application execution to drive node selection.

This paper addresses only a part of the challenge of employing a performance skeleton based approach to node selection. The results presented are restricted to sharing of communication resources only. We assume that all available computation nodes have the same available computation capacity but communication properties of the network links connecting the nodes are varying. Hence, node selection is based on bandwidth considerations only. In this scenario, a performance skeleton needs to be faithful to the original application in terms of communication behavior only. Hence, in order to be more accurate, we will refer to them as *communication skeletons* in this paper.

A communication skeleton is a short running program, and that is the key to keeping the run-time overhead of this approach acceptable. However, the number of possible groups of nodes that are candidates for application execution grows combinatorially with the total number of available nodes. Hence, it is not practical to execute even a short running communication skeleton on every candidate group of nodes. Therefore, we employ a separate procedure to select a set of candidate node groups from all available nodes. This algorithm is based on the information about the network status obtained from network measurement tools and the information about the communication pattern of an application, which is computed in a preprocessing phase. The final group of execution nodes is selected based on the execution time of the performance skeleton program on the candidate node groups.

This paper is organized as follows. The node selection framework is described in section 2. Section 3 describes our prototype implementation and results from experiments to validate the node selection procedure. Section 4 explains the capabilities and limitations of our approach and implementation, and discusses ongoing and future work. Section 5 contains conclusions.

2 Node selection framework

We first outline the main steps and components of the node selection framework.

The first two steps are performed ahead of time, once for each application.

1. Identify the main communication patterns of the candidate application.

2. Construct the communication skeleton of the application.

   The following subsequent steps are performed at the time the application has to be scheduled for execution.

3. Obtain current network status information.

4. Identify a small set of candidate node groups for execution by employing a node selection algorithm based on the network status and application's communication pattern.

5. Execute the communication skeleton program on each candidate group of nodes. Select the node group with the lowest execution time to schedule the application.

We now discuss each of the above steps in more detail.

2.1 Identification of communication pattern

The node selection framework relies heavily on the knowledge of the communication patterns in the application that has to be executed. These are captured by executing the application in a preprocessing phase on a controlled testbed and monitoring the message traffic between nodes. The procedure has been discussed in detail in [13, 15] in a related context. The methodology completely relies on system monitoring on the testbed while the application is executing and does not require application knowledge or access to the source code. The communication structure of NAS benchmark programs inferred from such runtime measurements is illustrated in Figure 2.

2.2 Communication skeleton program

The communication skeleton of an application is a synthetically generated program that replicates the dominant communication patterns employed by the application. As stated earlier, the size and pattern of the messages exchanged by the nodes executing an application are inferred by monitoring execution on a controlled testbed. Automatic construction of skeletons from this information is an important long term goal of our research. However, for the results presented in this paper, manually constructed communication skeletons were employed. A program that performs a set of representative message exchanges along the communication routes used by the application qualifies as a communication skeleton of the application.

2.3 Network status information

Our network status measurement module employs Network Weather Service [19], a freely available distributed resource monitoring system. NWS gathers system level resource information, such as CPU load and available bandwidth, for network connected compute nodes. We employ NWS to measure the available bandwidth between all nodes that can be used to execute an application. This step yields a graph with compute nodes as graph nodes and available bandwidth between them as graph edges. We will refer to this graph as the network status graph.

2.4 Node selection algorithm

The first step in the process of node selection is a procedure that analyzes the network status graph to choose a set of “good” candidate groups of nodes for application execution. Another input to the node selection procedure is the application structure, basically the number of nodes required to execute the application and pairs of nodes that communicate in the main data exchange patterns. The objective of this algorithm is to determine the group of nodes for which the minimum of the available bandwidth between communicating nodes is maximized. The reason for choosing this particular criterion is that the time for completion of a collective communication step in parallel programs is typically determined by the lowest bandwidth communication path rather than the average available bandwidth on communication paths.

This communication pattern based algorithm for node selection is presented in Figure 1. The algorithm is similar to the one that was introduced by Subhlok et. al. in [14], but with one important difference. The algorithm in Figure 1 attempts to optimize performance over network paths that are utilized by an application, while the algorithm in [14] treated all network paths connecting executing nodes equally. For example, in the algorithm in Figure 1, if the main application communication pattern is an all to all data exchange between computing nodes, then the network path between each pair of nodes is optimized. However, if the main communication pattern takes the form of a one dimensional ring, then only the paths composing the ring are considered for optimization.

We informally explain the node selection algorithm stated in Figure 1. Suppose the goal of the algorithm is to select m optimal nodes. The algorithm starts with the network status graph and repeatedly removes the edge with the minimum available bandwidth from the graph. At every step, the algorithm verifies that there are m nodes that are connected in a way that satisfies the communication pattern of the application. (e.g., if the communication pattern is a ring, there must be a path from one node to another such that a ring can be completed.) A path from one node to another can include network routers but not other computation nodes. When removing the minimum available bandwidth edge leads to a situation where m such nodes cannot be found, then the algorithm stops. The last step is reversed and a set of m nodes is selected.

The algorithm as presented in Figure 1 selects a single group of optimal nodes, but our framework is based on selection of a set of candidate node groups. In practice, the algorithm is easily modified for usage in our framework by backtracking the last few edge deletions and selecting all feasible node groups at that point.

2.5 Final node selection with communication skeletons

For final node selection, the communication skeleton program is executed on each group of candidate nodes selected by the node selection algorithm described above. The group of nodes on which the communication skeleton achieves the best performance is selected for application execution. An important consideration in this step is to not execute the communication skeleton concurrently on intersecting groups of nodes since execution on one group of nodes is likely to impact performance on other groups. Note that the communication skeleton program is short running, typically a few seconds long, and hence this stage is not likely to make a significant impact on the turnaround time of an application.

Input: A connected network status graph G. An application pattern graph A with m compute nodes representing the number nodes needed by the application and the application communicating pattern. That is, there is an edge between a pair of graph nodes in A if the link between the corresponding application nodes is included in the main application communication pattern. Assume that the number of computed nodes in G is at least m.

Output: A graph M containing m nodes that represents a mapping of the application to the compute nodes that maximizes the minimum bandwidth between any pair of communicating nodes as represented in A.

```
1. $M = null$

2. Attempt to find a subgraph $newM$ of $G$ such that there is a path between any two
   nodes of $newM$ if there is an edge between the corresponding nodes of $A$. If no such
   graph exists, set $newM = null$.

3. If ($newM == null$)
       return ($M$)
   Else
       $M = newM$

4. Remove the edge with the minimum available bandwidth from $G$.
   Goto Step 2.
```

Figure 1. Algorithm to select a set of nodes in a network status graph in order to maximize the minimum available bandwidth between any pair of communicating nodes based on a given application communication pattern graph.

3 Experiments and results

A prototype of the node selection framework discussed in this paper was implemented and validated on a network testbed. We first describe the experiments performed and then discuss the results.

3.1 Experimental setup

The testbed for the experiments is a compute cluster composed of 10 Intel Xeon dual CPU 1.7 GHz machines connected by 100Mbps Ethernet links and a full crossbar switch. All experimental results are based on the MPI implementation of the NAS Parallel Benchmarks [2, 16]. The codes used are BT (Block Tridiagonal solver), CG (Conjugate Gradient), IS (Integer Sort), LU (LU Solver), MG (Multigrad) and EP (Embarrassingly parallel). All programs are compiled using GNU `g77`, (Fortran) compiler except IS, which is compiled with the `gcc` (C) compiler. The MPICH implementation of MPI is used. The bandwidth between computation nodes was managed with the Linux advanced networking *iproute2* [1] in order to simulate limited bandwidth availability due to competing network traffic. *iproute2* works by intercepting the network packets and passing them through artificial queues to simulate bandwidth limitations.

3.2 Building communication patterns and communication skeletons

In order to make an application "ready" for automatic node selection, the main communication patterns have to discovered and a communication skeleton program has to be created in a preprocessing phase. For the NAS benchmark programs included in this study, the basic communication patterns were derived by execution on a dedicated testbed with system level monitoring of network traffic. We will skip the details of these measurements but they are discussed in [15, 17]. The results are illustrated in Figure 2.

The next objective is to construct the communication skeletons. The NAS benchmarks are available in several sizes labeled Class S,W,A,B,and C, in increasing order of the size of data structures and execution time. Class S benchmarks run within a few seconds on a small cluster,

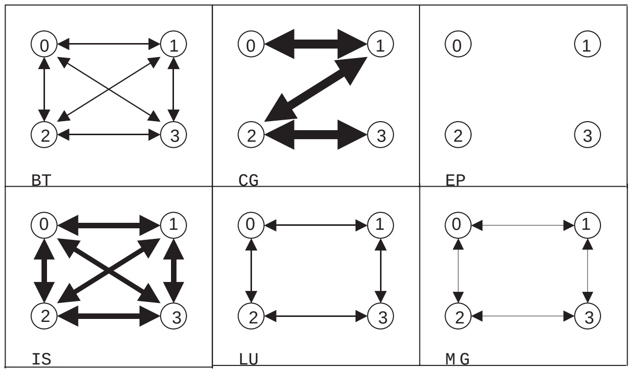

Figure 2. Dominant communication patterns during execution of NAS benchmarks. The thickness of the lines reflects the generated communication bandwidth.

while Class C benchmarks require a fairly large system to run at an acceptable speed. We chose class A benchmarks as the target applications to optimize. We also chose the corresponding class S benchmarks as the communication skeletons for the class A benchmarks, since they closely resemble each other and are likely to be very good skeleton programs. Our longer term goal in this research is to automatically construct performance skeletons, So, clearly, we are "cheating" by simply using a good skeleton program that happens to be available in this case. The reason is that we did not want the results to be impacted by the quality of the skeletons that we constructed since automatically building good skeletons is an open research problem that is not the focus of this paper. Hence, the results we obtained could be labeled as optimistic. However, based on other ongoing research, it is our firm belief that, in the near future, it will be possible to automatically generate skeletons of the quality that we have used for our experiments.

3.3 Automatic node selection

In order to evaluate node selection in the presence of network traffic, experiments were performed with varying available bandwidth caused by simulated network traffic. The available bandwidth on the network links connecting the computation nodes was controlled in the following manner. At any given time, every network link was assumed to be shared by a varying number of other traffic streams. If S streams are already sharing a network link, the bandwidth available to our application with fair sharing is assumed to be $1/(S + 1)$. Every 30 seconds, one traffic stream would randomly enter or leave the system, with a resultant increase or decrease in the available bandwidth on the affected link. The bandwidth, however, was never allowed to go below 10Mbps and cannot exceed the link capacity of 100Mbps. Based on the above simulation model, the actual bandwidth was controlled with the *iproute2* toolset.

Each NAS benchmark program was executed repeatedly on 4 nodes selected by our prototype node selection module based on the framework presented. NWS was employed to measure the available bandwidth between pairs of compute nodes on the network and build a network status graph. The node selection algorithm presented in Figure 1 was used to select the best three groups of nodes every time a benchmark program had to be scheduled and executed. Subsequently, the corresponding communication skeleton was executed on each of the three groups of nodes, and the group on which it performed the best was the selected node group. The benchmark program was then executed on those nodes and the execution time was measured and compared to the execution time on a dedicated testbed. For comparison, the procedure was also performed with two other node selection methods. The three node selection procedures that were evaluated and compared against each other are as follows:

1. **Pattern based:** The framework presented in this paper.

2. **All-all:** The nodes were selected using the network information, on the basis of maximizing the minimum available bandwidth between any pair of selected nodes. This approach requires a detailed analysis of the network status graph, but does not use any application specific information such as the communication pattern, and does not employ communication skeletons.

3. **Random:** Nodes were selected at random for reference.

The performance achieved by the benchmark programs on nodes selected by each of these methods was measured. The experiments were repeated a large number of times to get statistically meaningful results. For each benchmark program, the average execution time with each node selection procedure was computed and compared to the execution time of the same benchmark on a dedicated testbed with full bandwidth available on all links. The average slowdown due to link sharing for each benchmark program and each node selection procedure is presented in Figure 3.

3.4 Results

We observe from Figure 3 that each benchmark program performs significantly better when pattern based node selection is employed as compared to random node selection. On average, the percentage slowdown with random node selection is 40%, while that with pattern based node selection

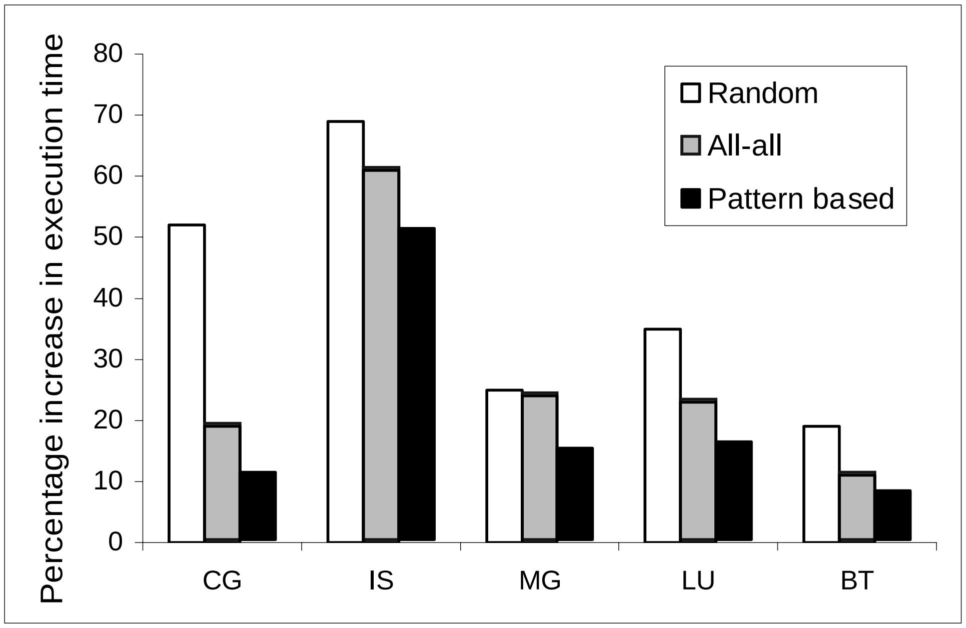

Figure 3. Percentage increase in the execution time of the NAS benchmarks due to network sharing for different node selection methods.

is around 20.2%. Hence, under these particular simulated conditions, the slowdown due to competing network traffic is reduced by half with good, application specific, node selection.

The node selection procedure labeled "all-all" can be considered a state of the art approach to node selection, but one that does not use the new concepts introduced in this paper. In the all-all method, the basis for node selection is maximizing the minimum available bandwidth between every pair of selected nodes, as described in [14]. The method entails a detailed analysis of the network status graph, but no consideration is given to the application communication structure, and communication skeletons are not used. The average slowdown with all-all node selection is 27.6% versus 20.2% for the pattern based framework. Hence, in our experiments, the pattern based approach to node selection reduces the slowdown due to link sharing by roughly a quarter as compared to a good approach that does not consider application communication patterns.

We observe that the general percentage slowdown as well as the relative performance with different node selection procedures varies dramatically across the programs in the NAS benchmark suite. EP benchmark is not included in the graph as it has no communication, and hence its performance is unaffected by the changes in available bandwidth and does not depend on the node selection procedure employed. The CG and IS benchmarks show the greatest percentage increase in execution time with random node selection. We observe from Figure 2 that these benchmarks are the most bandwidth hungry of the suite, which is the reason they are most affected by a reduction in the available bandwidth.

The pattern based scheme performs better than the random and all-all schemes for every application but there are significant differences. The maximum improvement in performance is observed for the CG benchmark. We speculate that the reason is that only 3 pairs of nodes communicate in CG as shown in Figure 2. A smart node selection procedure has a better chance of finding a relatively small number of "good" network paths as compared to finding good paths between every pair of selected nodes. This translates to finding 3 good paths versus 6 good paths for 4 nodes. Further, as mentioned earlier, CG is among the most communication intensive programs in the suite, and therefore, its performance is most sensitive to network path selection.

Another observation is that the relative improvement with pattern based node selection, as compared to all-all node selection, is lowest for IS and BT benchmarks. This is not surprising since the main communication pattern in IS and BT benchmarks is an all to all data exchange. Hence, the analysis of network status graph is identical for pattern based and all-all procedures and the difference is only due the use of communication skeletons in the pattern based framework.

The broad conclusion from these experiments is that the pattern based approach to node selection offers considerable improvement in expected performance over random node selection and all-all node selection procedures, but the extent of improvement is strongly dependent on the application characteristics. We should also caution that these are limited experiments and the results will also strongly depend on the network and system characteristics.

4 Discussion

This research employs application characteristics to drive the process of automatically selecting network nodes to execute an application. Specifically, we suggest a two step process for node selection. In the first step, a network status map is matched to the application communication structure to obtain a set of potential node groups for execution. Clearly, better node selection decisions can be made if the procedure is sensitive to application characteristics. In the second step, an application communication skeleton is executed on every candidate group of nodes to decide which group offers the best potential performance. This step is intended to eliminate the various inaccuracies in estimating the communication performance an application can expect on a group of network nodes.

The focus of this paper has been entirely on communication characteristics. We only consider variations in network availability and base node selection on communication capacity. In practice, computation and synchronization considerations are equally important. Computation nodes may have competing loads, and impact of a slowdown in one node in the system may get magnified because of data and control dependencies. In related work, we have addressed the problem of performance estimation with shared nodes and links [17]. However, effective integration and validation of these techniques in a node selection system remains a challenge.

We have only employed communication skeletons, that are a special case of performance skeletons. Construction of complete performance skeletons also includes computation and synchronization considerations. Perhaps the most critical limitation of our system is that the communication skeletons have to be constructed manually. It is not difficult to automatically construct a program that concisely reproduces the measured communication pattern of an application. However, our real goal is automatic construction of general performance skeletons. A performance skeleton should mirror the application it represents in all respects. For example, the computation to communication ratio, synchronization patterns, memory access patterns, fraction of different types of instructions, message exchange patterns, should all be closely correlated between an application and its performance skeleton. The goal is that the relative behavior of the application and performance skeleton should be similar under any computation environment and under all network conditions. And yet the performance skeleton is expected to execute for a very short time. Note that a performance skeleton cannot be just the beginning part of the application itself since application behavior changes over time and the performance skeleton is expected to capture the cumulative application activities over the full duration of execution. Clearly, automatically constructing performance skeletons is a major challenge, and it is also a key long-term goal of this research. This paper focuses only on demonstrating the value of performance skeletons for application scheduling.

The prototype implementation and results described in this work are essentially a “proof of concept”. We point out the most significant limitations of our implementation and experiments. The prototype node selection tool automatically determines the best nodes for execution and schedules the application on those nodes. However, some of the steps in the preprocessing of the applications, to enable them for automatic node selection, are manual. We have conducted experiments on a small compute cluster with the bandwidth controlled to simulate network sharing. More experimentation on larger clusters and grid environments is necessary to evaluate this approach rigorously. Our system currently works only for MPI message passing applications but is not fundamentally limited to any programming model. The NAS benchmark programs used in this research represent a variety of applications in parallel computing, but each benchmark focuses on a single core scientific algorithm. Full applications, in contrast, often employ multiple different types of computations in different phases. This certainly adds additional complexity to node selection that is not evaluated in this work. Overall, we believe that our results are relevant and meaningful, even though there is significant room for more experimentation and better evaluation and validation.

5 Conclusions

This paper makes a case for employing application knowledge to node selection in shared cluster and grid environments. We demonstrate how the communication pattern of an application is exploited to discover good compute nodes and network paths for execution. One of the major problems in automatic node selection for network environments is the cost and accuracy of network usage information. We propose application communication skeletons as our solution approach. With the use of this method, approximate network information is used to get good candidate node groups for execution, and actual execution of skeletons is used to make the final choice. This largely eliminates the potential for poor choices due to inaccurate network information since a small slice of actual execution is performed before assignment of nodes to an application.

We have developed a prototype node selection framework and present results from a network testbed that simulates varying bandwidth availability between pairs of nodes available for execution. The results clearly demonstrate that node selection based on this framework is a large improvement over random node selection, and also a clear improvement over state of the art methods that do not employ application knowledge. While our prototype implementation and experiments are limited in scope, they clearly demonstrate the potential of automated node selection and scheduling based on an application's communication pattern. This paper is a significant step towards general resource scheduling that employs broader application knowledge including computation and synchronization information and general performance skeletons.

6 Acknowledgments

This research was supported in part by the Los Alamos Computer Science Institute (LACSI) through Los Alamos National Laboratory (LANL) contract number 03891-99-23 as part of the prime contract (W-7405-ENG-36) between the DOE and the Regents of the University of California. Support was also provided by the National Science Foundation under award number NSF ACI-0234328 and the University of Houston's Texas Learning and Computation Center.

We wish to thank other current and former members of our research group, in particular, Mala Ghanesh, Amitoj Singh, Sukhdeep Sodhi and Shreenivasa Venkataramaiah, for their contributions to this research.

References

[1] W. Almesberger. Linux network traffic control — implementation overview. White Paper, April 1999. Available at ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.

[2] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report 95-020, NASA Ames Research Center, December 1995.

[3] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96*, Pittsburgh, PA, November 1996.

[4] P. Bhatt, V. Prasanna, and C. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.

[5] J. Bolliger and T. Gross. A framework-based approach to the development of network-aware applications. *IEEE Trans. Softw. Eng.*, 24(5):376 – 390, May 1998.

[6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the grid. In *Supercomputing 2000*, pages 75–76, 2000.

[7] I. Foster and K. Kesselman. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[8] A. Grimshaw and W. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[9] M. Litzkow, M. Livny, and M. Mutka. Condor — A hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems*, San Jose, California, June 1988.

[10] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.

[11] B. Lowekamp, D. O'Hallaron, and T. Gross. Direct queries for discovering network resource properties in a distributed environment. *Cluster Computing*, 3(4):281–291, 2000.

[12] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *9th Heterogeneous Computing Workshop*, pages 3–16, 2000.

[13] A. Singh and J. Subhlok. Reconstruction of application layer message sequences by network monitoring. In *IASTED International Conference on Communications and Computer Networks*, November 2002.

[14] J. Subhlok, P. Lieu, and B. Lowekamp. Automatic node selection for high performance applications on networks. In *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 163–172, Atlanta, GA, May 1999.

[15] J. Subhlok, S. Venkataramaiah, and A. Singh. Characterizing NAS benchmark performance on shared heterogeneous networks. In *11th International Heterogeneous Computing Workshop*, April 2002.

[16] T. Tabe and Q. Stout. The use of the MPI communication library in the NAS Parallel Benchmark. Technical Report CSE-TR-386-99, Department of Computer Science, University of Michigan, Nov 1999.

[17] S. Venkataramaiah and J. Subhlok. Performance prediction for simple CPU and network sharing. In *LACSI Symposium 2002*, October 2002.

[18] J. Weismann. Metascheduling: A scheduling model for metacomputing systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.

[19] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The Network Weather Service. In *Proceedings of Supercomputing '97*, San Jose, CA, Nov 1997.

[20] S. Zhou. LSF: load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the Workshop on Cluster Computing*, Orlando, FL, April 1992.