Automatic Node Selection for High Performance Applications on Networks

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

Peter Lieu and Bruce Lowekamp
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{lowekamp,pjl}@cs.cmu.edu

Abstract

A central problem in executing performance critical parallel and distributed applications on shared networks is the selection of computation nodes and communication paths for execution. Automatic selection of nodes is complex as the best choice depends on the application structure as well as the expected availability of computation and communication resources. This paper presents a solution to this problem for realistic application and network scenarios. A new algorithm to jointly analyze computation and communication resources for different application demands is introduced and a framework for automatic node selection is developed on top of *Remos*, which is a query interface to network information. The paper reports results from a set of applications, including Airshed pollution modeling and magnetic resonance imaging, executing on a high speed network testbed. The results demonstrate that node selection is effective in enhancing application performance in the presence of computation load as well as network traffic. Under the network conditions used for experiments, the increase in execution time due to compute loads and network congestion was reduced by half with node selection. The node selection algorithms developed in this research are also applicable to dynamic migration of long running jobs.

1 Introduction

Networked computers are the platform of choice for a wide range of parallel and distributed applications as they are ubiquitous and cost-effective. Further, emerging performance critical applications often contain components that require execution on geographically distributed and heterogeneous systems. Many network applications are not tied to specific hosts, but can choose execution nodes from a pool of available nodes. Parallel scientific applications often select a subset of nodes from one or more compute clusters. Client-server applications may have a choice of machines on which to run a client, or select from a set of distributed servers. Network conditions change continuously due to sharing of resources, and computation load, as well as congestion on network nodes and links, can increase the application response time dramatically. Therefore, a key challenge in exploiting such a distributed computation environment for high performance applications is good node selection.

The goal of this research is to enable parallel and distributed applications to automatically select the best available computation nodes on the network for execution. The major components of a framework to achieve this goal are the following:

1. An interface for applications to specify their computation and communication structure and requirements.
2. A mechanism to query the status of the network, including the current load on the computation nodes and traffic on the communication links.
3. Procedures for node selection based on application requirements and network status.

A uniform external interface for specification of application behavior is an important component of the node selection framework as it allows unmodified applications to use automatic node selection. The ability to get accurate dynamic network information is a major challenge and an important requirement for automatic node selection. Fortunately this topic has been the focus of several recent research efforts including the Remos system developed by our group at Carnegie Mellon. Remos provides applications with an

Appears in the Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '99)

interface to their execution environment and its important features are outlined in the paper.

The central contribution of this paper is a new set of procedures for node selection on heterogeneous networks. The algorithms use computation, communication, or a combination of the two, as the optimization criterion, and can be tuned to a variety of application goals. We have implemented a complete framework for node selection and performed a set of experiments to validate the importance and accuracy of the system. Experimental results are presented for a suite of applications consisting of FFTs, Airshed pollution modeling, and magnetic resonance imaging.

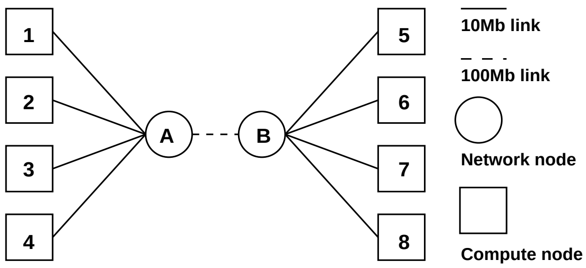

Figure 1: Remos graph representing the structure of a simple network.

2 Framework for node selection

Our framework for node selection includes an interface for applications to specify their execution requirements, and the Remos interface to network information. We briefly describe these components here, and will examine the node selection procedures in detail in the next section.

2.1 Application specification interface

The structure and processing requirements of an application are the driving inputs for automatic node selection. This application level knowledge can be difficult or impossible to discover automatically, hence it is important to allow the application or the programmer to specify these characteristics. The information that is transferred through this interface includes the following: the number of nodes required for execution, the nature of main computation and communication patterns (e.g. all-to-all or master-slave), relative priority of communication and computation, different node groups within an application (e.g. client and server groups), specific requirements of different groups (e.g. a server may be compiled only for Alpha architecture or must run on some specific machines). While this API is an important component of our automatic node selection framework, it is not the main topic of this paper and we will not discuss it further.

2.2 Remos API

Remos is an API designed to support the development of network-aware parallel and distributed applications on diverse network architectures. The API is in the form of a query interface, and is portable across network architectures and installations. The cost that an application pays in terms of runtime overhead is low and directly related to the depth and frequency of its requests for network information.

Remos API exports network information at two levels of abstraction: *flow queries* and *logical network topology*. Flow queries provide specific information like available bandwidth between pairs of nodes and account for sharing of network links by multiple flows. The logical network topology presents a functional snapshot of the relevant part of the network, including the traffic on communication links and switches. An example topology graph is shown in Figure 1. It is important to note that the logical topology graph contains structural network information that cannot be captured by measurements between pairs of compute nodes, and this research exploits this extra information to develop faster and more accurate node selection procedures.

Remos information includes load on the compute nodes as well as the capacity, utilization and latency of network links. Remos can be queried for information based on a fixed window of history, current network conditions, or an estimate of the future availability. The local area implementation of Remos is based on *SNMP* processes on network nodes and entails a very low overhead. More details of Remos design and implementation are available in [15] and a full API is described in [4].

3 Node selection procedure

We first describe the structure of the network topology graph, which is the basis for node selection. We present a set of algorithms to solve the simple cases of node selection, and then discuss the extensions and restrictions in the context of more general network and application environments.

3.1 Network topology graph

The network topology graph is an undirected connected graph $G(n)$ containing n nodes. A node in this graph is a *compute node* or a *network node*. A compute node represents a processor that is available for computation, while a network node represents a network device used for routing communication. The edges in the graph represent communication links between the nodes.

We should point out that the Remos API and our node selection procedures support directed edges between nodes that represent dedicated unidirectional communication links. However, for simplicity of presentation, we are starting with the assumption that the graph is undirected, and will explicitly discuss extensions for directed graphs later in this section.

A function $cpu(i)$ is defined for each compute node n_i and it represents the fraction of the computation power of the node that is available to an application. The cpu function is computed from the load average on a processor as follows:

$$cpu = 1/(1 + loadaverage)$$

The justification is that the load average represents the number of active processes, and the processor will be equally shared by those processes and the user application process. That is, we are implicitly assuming that all jobs have equal execution priority.

Functions $maxbw$ and bw are defined for each pair of nodes connected by an edge. $maxbw(i, j)$ is the peak bandwidth between the nodes n_i and n_j, while $bw(i, j)$ is the corresponding currently available bandwidth. We also define $bwfactor$, as the fraction of the peak bandwidth that is available, i.e.:

$$bwfactor = bw/maxbw$$

3.2 Fundamental node selection algorithms

We present algorithms for node selection for maximizing available computation capacity, for maximizing available communication capacity, and with different weightage to computation and communication. We make a number of assumptions here, in part to simplify the presentation. We assume that the network topology graph is acyclic, i.e., there is only one path for data to travel between a pair of nodes. We also assume that the network environment is homogeneous, i.e., all computation nodes are identical and all communication links have the same capacity. The basis used for communication optimization is only bandwidth. We will separately discuss these and other assumptions as well as algorithmic changes required to relax them.

Maximize computation capacity

For a homogeneous system, node selection for maximizing available computation capacity can be done effectively by simply choosing the nodes with the least amount of load. In our graph terminology, if an application requires m nodes, we simply select the m nodes with the highest values of the cpu function.

Maximize communication capacity

The criterion used for node selection for maximizing available communication capacity is to *maximize the minimum of the available bandwidth between any pair of selected nodes*, i.e., to minimize the bottleneck communication path. Note that we are assuming that all communication links have equal capacity, but the *available bandwidth* on a link is a dynamically varying quantity, and will, in general, be different for different links. An algorithm that selects a set of nodes to optimize available communication capacity based on the above criterion is outlined in Figure 2.

This algorithm is based on the following simple observation. For a set of connected nodes in an acyclic topology graph, the least bandwidth between any pair of nodes in the set cannot be less than the lowest edge bandwidth in the graph. Hence, by repeatedly removing the minimum available bandwidth edge and testing if enough connected nodes exist in the graph, the node-set that maximizes the minimum available bandwidth between any pair of nodes is obtained. In terms of the algorithm description in Figure 2, the size l of the largest connected component L of graph G will keep decreasing as the minimum bandwidth edges are removed from the graph. Eventually this size will become less than m, the number of connected nodes required for execution. At that point, a set of m nodes picked from the previous L (which are in the current set M) are selected as an optimal set of nodes for execution.

Balanced communication and computation optimization

We present an algorithm that selects a set of nodes to obtain *the maximum fraction of compute and communication capacity that can be achieved simultaneously*. The fraction of peak computation capacity available to a set of nodes is the minimum of the fractional cpu capacities (cpu) available to the nodes in the set. Similarly, the fraction of peak communication capacity available to a node set is the minimum of the fractional peak bandwidth ($bwfactor$) available between pairs of nodes in the set. Alternately stated, the (fractional) computation and communication capacities for a set of nodes are determined by the most loaded node and the path with the maximum traffic, respectively. We have assumed that all compute nodes and network links have the same capacity, and we are further assuming that the application is weighting communication and computation equally. Both these assumptions are for simplicity of presentation and their relaxation is discussed separately. The algorithm is stated in Figure 3.

The goal of this algorithm is to select a set of nodes such that the minimum of the minimum fractional cpu capacity and the minimum fractional bandwidth capacity is maximized. The algorithm at first selects m nodes to optimize computation capacity without regard to communication. Subsequently, the algorithm repeatedly removes the graph edge corresponding to the lowest fractional bandwidth ($minbw$) in an attempt to improve the minimum fractional bandwidth capacity, but this is likely to reduce the minimum fractional cpu capacity ($mincpu$) in the graph. At every step,

Input: A connected logical topology graph $G(n)$. Number of nodes required for execution m and given that number of compute nodes in G is at least m.
Output: A set M containing m nodes that maximizes the minimum bandwidth between any pair of selected nodes.

1. M = $\{$Any m compute nodes in $G\}$
2. Remove the edge with the minimum available bandwidth (lowest *bw*) from G.
3. Find the largest number l of connected compute nodes in G, and let the corresponding connected graph component be L.
4. If $(l > m)$
   M = $\{$Any m compute nodes in $L\}$
   Goto Step 2.
5. M contains an optimal set of m nodes.

Figure 2: Algorithm to select a set of nodes to maximize the minimum available bandwidth between any pair of nodes

$minresource$, which is the minimum of the "minimum fractional bandwidth" and "minimum fractional cpu capacity", is compared to its previous value, and the process continues until this $minresource$ keeps increasing. The algorithm stops when removing this communication bottleneck link does not lead to a higher low for fractional capacities $minresource$ (because of reduced minimum fractional cpu capacity) or if a connected graph with sufficient number of compute nodes is no longer available. This is a simple greedy algorithm that is based on the fact that the minimum bandwidth available on any path in a graph cannot be lower than the minimum bandwidth edge in the graph.

Computation complexity

For an acyclic connected topology graph G with n nodes, it is easy to see that selection of the set of nodes with maximum computational capacity simply takes $O(n)$ time. We now consider the algorithms for maximizing computation capacity and balanced computation and communication optimization. For both these algorithms, we observe that the number of iterations is bounded by n and the work in each iteration consists of linear graph procedures that is bounded by $O(n)$. Hence the total complexity of finding the optimal set of nodes is $O(n^2)$. Note that the number of nodes n includes the computation nodes in the network as well as the communication nodes like switch nodes.

In our experiments, the computation time of these algorithms has been insignificant in comparison with the execution times of the applications that they were applied to, but our experience is limited to small to moderate size testbeds. The computation time of the algorithms is a potential bottleneck for very large testbeds and it may be necessary to use heuristics and knowledge of the network configuration to speed up the node selection process. We have not encountered or addressed this problem in our research and we shall not discuss it further.

3.3 Generalized node selection

We discuss some important extensions to the node selection algorithms. The presentation is centered around the balanced communication and computation optimization algorithm as the others are a special case of this algorithm.

Heterogeneous links and nodes: The assumption that all links have the same bandwidth capacity is not necessary and was made for simplicity of presentation. If multiple capacity links exist, a reference link has to be specified for balancing against computation. e.g., if the network contains 100Mbps and 155Mbps links, the reference link will determine if 50% available bandwidth is 50Mbps or 77.5 Mbps for the purpose of balancing computation and communication. Similarly, the solution procedures can be modified relatively easily to handle nodes with different computation capacities. The requirements are that the relative computation capacities of the nodes must be known and a reference node type must be specified.

Prioritization of computation and communication: The balanced computation and communication algorithm attempts to simultaneously deliver the highest fraction of peak communication and computation capacity. The procedure is easily modified to prioritize the optimization of one by a given factor. For example, if computation was prioritized by a factor of 2, 50% CPU availability would be considered equivalent to 25% availability of communication paths.

Input: A connected logical topology graph $G(n)$. Number of nodes required for execution m and given that the number of compute nodes in G is at least m.
Output: Optimal node set M containing m nodes based on maximizing the minimum of available fractional compute and communication capacities.

1. M = $\{m$ nodes with maximum available cpu capacity in $G\}$
   $mincpu$ = minimum available cpu capacity (cpu) among the nodes in M
   $minbw$ = minimum available fractional bandwidth ($bwfraction$) among the edges in G
   $minresource$ = $min(mincpu, minbw)$

2. Remove an edge corresponding to minimum bandwidth $minbw$ in G
   $newsetflag = FALSE$

3. Find all connected components of the current graph. For each connected component L with at least m nodes:
   $newM$ = $\{m$ nodes with maximum available cpu capacity (cpu) in $L\}$
   $mincpu$ = minimum available cpu capacity among the nodes in $newM$
   $minbw$ = minimum available fractional bandwidth ($bwfraction$) among the edges in L
   $newminresource$ = $min(minbw, mincpu)$
   if ($newminresource > minresource$)
   $minresource = newminresource$
   $M = newM$
   $newsetflag = TRUE$

4. If ($newsetflag == TRUE$) Goto Step 2.

5. M contains an optimal set of m nodes.

Figure 3: Algorithm to select a set of nodes to maximize the minimum available fractional compute and communication capacities

Fixed computation and communication requirements: The procedures can be adapted to satisfy a fixed bandwidth requirement (e.g. a minimum of 50Mbps between any selected nodes) and maximize processor availability under that constraint, and vice versa. The algorithm structure is not modified and new constraints are added that define eligible node sets.

Cycles in network topology: The algorithms assume that the network graph is acyclic, but network topologies often contain cycles, implying multiple paths between nodes. However, networks typically use static routing implying that a fixed path is actually taken for all communication between a pair of nodes. Our algorithms are directly applicable in this case, but are not suitable for dynamic routing networks.

Independent and shared network links: We have assumed that the same shared communication fabric is used for carrying data in either direction on a link between a pair of nodes in the network, which allows us to represent the network with an undirected graph. However, this is often not true, i.e., often a pair of nodes is connected by two distinct links to carry data in each direction. Such bidirectional links are handled by Remos that provides the topology graphs, and by our node selection procedures. The available capacity of the a bidirectional link is taken to be the minimum of the available capacities in each direction. The analysis details are modified to take this into account, but the basic algorithm is not changed.

Dynamic migration: The solution procedure can be applied directly to the problem of dynamic migration to avoid network congestion and busy nodes, and some preliminary experience is discussed in [11]. One important consideration is that the load and traffic caused by the application itself must be captured separately as it is not due to a competing process.

3.4 Limitations

We discuss some important limitations of our node selection procedures.

Simultaneous traffic streams: We have computed availability of bandwidth between any pair of nodes independently. However, if multiple communication operations in an application happen at exactly the same time and share a network link, then one or both may achieve a lower effective bandwidth. Remos is capable of handling the sharing of network links, but it is often impossible to predict the relative timing of communication operations in an application. Hence, this is a difficult problem that is not addressed by this research.

Latency and other considerations: Our node selection procedures are based on the load on compute nodes and available bandwidth between them. A number of other factors can affect application performance, some examples being latency on the links, and memory and disk availability on the compute nodes. Remos API includes this information and we plan to take these factors into consideration in future work.

Networks with reservations: We have implicitly assumed that the network environment assigns resources on *best-effort* basis to applications, but in practice many networks support explicit reservation of resources. Of course, if an application is executing with reserved resources, then the problem addressed in this research does not occur. However, if an application is executing on best-effort basis, while there may be other applications that use reservations, our methods are still applicable so long as the available resources can be computed and made known to the processor selection algorithms. In general, the concept of availability and sharing policy for resources is fundamentally changed with reservations, and hence this a complex topic that is beyond the scope of this paper.

Custom execution patterns: Our node selection procedures attach equal importance to all nodes and communication paths. However, this is not accurate for all applications. For instance, a client-server application may require that the node with the maximum available computation capacity be assigned to the server, and that only communication from the servers to the clients is significant. Our application interface allows description of such scenarios (and Remos has the relevant information), and we are currently investigating the algorithm extensions necessary to accurately handle a richer set of application patterns.

Variable number of execution nodes: We have assumed that an application executes on a fixed number of nodes. However, for many parallel applications, the exact number of nodes for execution can be decided at the time of invocation. The decision procedures developed in this research can be applied to the problem of finding the number *and* the set of nodes for execution, but do not solve the entire problem. These techniques have to be coupled with methods for performance estimation, for example [7, 19], to address this broader problem for networked systems.

4 Experiments and results

The node selection procedures presented in this paper have been implemented and in use on a networking testbed at Carnegie Mellon. To validate the algorithms, node selection was performed in the presence of realistic computation and communication loads. We describe the setup for experiments and then present results.

4.1 Network testbed

All experiments were performed on a part of a networking testbed at Carnegie Mellon, that is illustrated in Figure 4. The part of the testbed used for these experiments employs DEC Alpha compute nodes, Cisco routers, and 100 Mbps ethernet links with one 155Mbps ATM link (connecting the *gibraltar* and *suez* routers).

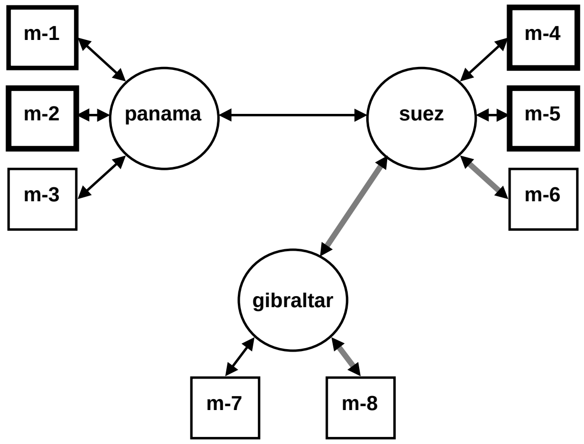

Figure 4: IP-based testbed for implementation and experiments. The compute nodes are DEC Alphas labeled *m-1* to *m-8*. Routers, labeled *panama*, *suez* and *gibraltar* are Cisco routers. All links are 100Mbps Ethernet links, except that the link between *gibraltar* to *suez* is a 155 Mbps ATM link. The figure also shows 4 nodes (with bold borders) that were automatically selected to avoid a traffic stream from *m-6* to *m-8*
.

4.2 Load and traffic generators

Procedures for automatic node selection should do a good job with realistic loads and traffic on the network. But it is virtually impossible to define what is *realistic*, as the load and traffic conditions vary dramatically in network environments. For the purpose of obtaining credible results, we used the results of recent research in characterizing resource usage patterns, and set parameters intuitively to reflect a testbed that is used primarily for data and compute intensive computations.

A synthetic compute intensive job was periodically invoked on every node. Processor load was generated using models developed by Harchol-Balter and Downey, whose measurements indicate Poisson interarrival times, with job duration determined by a combination of exponential and Pareto distributions [12]. Because we are interested in environments which support compute and data intensive computations, higher parameters were used for the load generators than would be used to represent typical interactive systems. Assuming that our target environment is a cluster or group of workstations in a single department, the workload distribution study by Harchol-Balter should be accurate, because their model was derived from observations in such an environment.

For generating network traffic, messages were periodically sent between random nodes. Message interarrival times were Poisson, with message length having a LogNormal distribution. The bulk of the research in network modeling has focused on internet-level traffic representation, rather than for local area networks. Although there are problems with using Poisson interarrival times for representing bulk traffic and some characteristic of aggregated traffic, it represents the interarrival times of the large high-speed data transfers we would be most concerned about in our target environment rather well [16, 17].

A full validation of the strengths and weaknesses of our techniques would require a large number of experiments with different network usage models and different parameters. While we are not at this stage in our experiments, we believe that a study with load and traffic generators that are realistic in some environments does establish the fundamental value of our node selection procedures.

4.3 Results

We employed the following 3 applications: 2D fast Fourier transform (32 iterations), Airshed pollution modeling (6 hour simulation) [22], and magnetic resonance imaging(*epi* dataset) [6, 9] to validate our decision procedures. Each application was executed several times with the computation load generator on, network traffic generator on, and with both generators on. Node selection was alternately made randomly and with our automatic node selection procedure. Our experience and previous results indicate that random node selection and node selection based on static network properties give virtually identical performance on a small testbed with all high speed links like ours [15], and hence the random selection results also apply to static node selection procedures. The results are presented in Table 1. Each measurement is the average of a number of executions spanning several hours. Since the activity on the network is changing continuously, a large number of measurements is necessary to have statistically relevant results.

We observe that for all three applications, the load and traffic generators significantly increase the execution time, as compared to the the time with no load (last column of the graph), and their combined effect is cumulative. The impact is fairly high for the FFT and Airshed programs (range of 300% with both generators on and execution on random nodes) but relatively modest for MRI(maximum of around 25%). The reason is that the FFT and Airshed programs are loosely synchronous parallel computations where any computation or communication step can become a bottleneck, while MRI uses a master-slave protocol for compute intensive regions that automatically adapts if a compute or communication step slows down.

In each of these cases, automatic node selection reduces the execution time significantly as compared to random node selection, specifically 8-14% for MRI, 16-23% for FFT and 32-35% for Airshed. We now focus on the increase in execution time due to traffic and load. When using random nodes and with both traffic and load generators on, the FFT time went up from 48 to 142.6 seconds (201%), Airshed from 150 to 530.2 seconds (253%) and MRI from 540 to 776 seconds (43.7%). Correspondingly, with automatic node selection, the increase in execution time was 145% for FFT, 103% for Airshed, and 23% for MRI. Making similar comparisons for other cases, we come to the result that **the increase in execution time due to traffic and/or load is approximately cut in half with automatic node selection**. While we should caution that this result is certainly not applicable to all applications or network conditions, it clearly demonstrates that our node selection procedures are effective in reducing the impact of link and processor sharing on applications.

4.4 Discussion

We have presented a set of results under just one model of computation load and communication traffic and with fixed parameters. However, since the models are probabilistic, a large number of measurements were necessary to obtain statistically relevant results. The results presented in this paper represent several days of execution on the testbed.

More experimentation is needed to address a number of questions, including the generality of our procedures as well as sensitivity of automatic node selection to load and traffic on one hand, and application length and characteristics on the other. Addressing these issues satisfactorily requires an amount of experimentation that we could not attain because of limited resources. More importantly, this points to the

Application		Execution Time with External Load and Traffic (seconds)						Reference Execution time on Unloaded Testbed
		Randomly selected Nodes			Automatically selected Nodes			
Name of Program	No of Nodes	Processor Load *seconds*	Network Traffic *seconds*	Load+ Traffic *seconds*	Processor Load *secs(% change)*	Network Traffic *secs(% change)*	Load+ Traffic *secs(% change)*	
FFT (1K)	4	112.6	80.3	142.6	82.6 (-23.8%)	64.6 (-19.5%)	118.5 (-16.7 %)	48
Airshed	5	393.8	281.3	530.2	254.0 (-35.2%)	188.5 (-32.9%)	355.1 (-33.0%)	150
MRI	4	683	591	776	594 (-12.7%)	571 (-8.4%)	667 (-14%)	540

Table 1: Performance in the presence of computation load and network traffic with automatically selected nodes and random nodes

importance of combining analytical and practical methods to achieve a degree of confidence in resource management methods that cannot be achieved by analysis or experimentation alone as both have their limitations.

5 Related Work

A number of systems for management of shared computation resources exist, some examples being Condor [14] and LSF(Load Sharing Facility). However, they primarily focus on the availability of computation nodes and do not take the communication constraints into account. Furthermore, our work targets node assignments to maximize application performance rather than system throughput.

More recently, resource management systems have been designed for large scale internet-wide computing, e.g., Globus [8] and Legion [10]. These systems provide support for a wide range of functions such as resource location and reservation, authentication, and remote process creation mechanisms. The focus of this work is node selection which is complementary to the contributions of these frameworks.

This research uses Remos [15] to measure the availability of network resources, and simply uses the most recent measurements as a forecast for the future. Research into forecasting of network performance [26] and availability of computation capacity [5] is orthogonal but relevant to this research.

Scheduling applications over wide-area distributed systems has attracted considerable attention. AppleS [1] is designed for application-centric scheduling of tasks over heterogeneous wide-area networks, and it relies on Network Weather Service [26] for resource information. Research with similar goals in the Legion framework is described in [25]. In the process of application scheduling, these systems also address the problem of selecting nodes for execution. However, the specific resource selection problem that we address is qualitatively different because we operate on the logical network topology provided by Remos, rather than the information on availability of communication resources between pairs of nodes. This is an important advantage as our algorithm can directly eliminate busy links in the network when selecting execution nodes. While the same result can be obtained by examining communication bandwidth between pairs of nodes, the availability of a logical network topology offers a more efficient and scalable solution.

Many application specific network measurement and adaptation systems have been developed, some examples being [13, 3, 21, 24]. An important goal of this research is to develop a framework that can be used by a large class of applications. A shared memory based approach to adaptive parallelism is explored in [18]. Node assignment and scheduling algorithms in the literature typically do not treat communication in realistic detail, but some recent exceptions are [23, 2]. Several runtime support systems have been developed for partitioning and scheduling computation and communication, an example being [20]. However, the primary job of theses systems is not node selection but application scheduling.

6 Concluding remarks

Automatic selection of network nodes for parallel and distributed programs is a hard problem and this paper introduces a solution framework with a new node selection algorithm. We have made a number of assumptions in order to develop a manageable solution, and certainly more research and experimentation is needed for a more general solution to this problem. However, we have obtained good results on real applications under a realistic load and traffic scenario. This is a tough and realistic way to validate this research, even though it does not establish the generality of the techniques. We specifically demonstrate that our load selection framework was effective in halving the effect of network congestion and machine sharing on application turnaround time. Hence, we believe that we have a good solution and this paper represents an important step towards making networked systems like workstation clusters and metacomputers a practical and attractive platform for performance sensitive applications.

This framework uses Remos for network status information, and further development of tools like Remos and Globus to manage and blend diverse information from wide-area and local-area networks is crucial for further progress. Continued research into forecasting future availability of network and computation resources is also important. Our framework has successfully used the state of the art in gaining information about networks and helps understand how better information will lead to better execution environments.

7 Acknowledgments

The Remulac and Darwin groups at Carnegie Mellon have built the infrastructure used in this research. Special thanks go to Peter Dinda, Thomas Gross, Nancy Miller, David O'Hallaron, Tim Newsome, Peter Steenkiste, and Dean Sutherland. We thank Bill Eddy, Kate Fissell, Greg Hood and Joel Welling of the *Fiasco* group for making the magnetic resonance imaging application available to us and providing excellent support for installation and experiments. We would also like to thank the reviewers for insightful comments that led to significant improvements in the paper.

This research is sponsored by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

References

[1] BERMAN, F., WOLSKI, R., FIGUEIRA, S., SCHOPF, J., AND SHAO, G. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96* (Pittsburgh, PA, November 1996).

[2] BHATT, P., PRASANNA, V., AND RAGHAVENDRA, C. Adaptive communication algorithms for distributed heterogrneous systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing* (Chicago, IL, July 1998).

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

[4] DEWITT, T., GROSS, T., LOWEKAMP, B., MILLER, N., STEENKISTE, P., SUBHLOK, J., AND SUTHERLAND, D. Remos: A resource monitoring system for network-aware applications. Tech. Rep. CMU-CS-97-194, Carnegie Mellon University, Dec 1997.

[5] DINDA, P. Statistical properties of host load in a distributed environment. In *Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers* (Pittsburgh, PA, May 1998).

[6] EDDY, W., FITZGERALD, M., GENOVESE, C., MOCKUS, A., AND NOLL, D. Functional image analysis software - computational olio. In *Proceedings in Computational Statistics* (Heidelberg, 1996), A. Prat, Ed., pp. 39–49.

[7] FAHRINGER, T., BASKO, R., AND ZIMA, H. Automatic performance prediction to support parallelization of Fortran programs for massively parallel systems. In *Proceedings of the 1992 International Conference on Supercomputing* (Washington, DC, July 1992), pp. 347–56.

[8] FOSTER, I., AND KESSELMAN, K. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications 11*, 2 (1997), 115–128.

[9] GODDARD, N., HOOD, G., COHEN, J., EDDY, W., GENOVESE, C., NOLL, D., AND NYSTROM, L. Online analysis of functional MRI datasets on parallel platforms. *The Journal of Supercomputing 11* (1997), 295–318.

[10] GRIMSHAW, A., WULF, W., AND LEGION TEAM. The Legion vision of a worldwide virtual computer. *Communications of the ACM 40*, 1 (January 1997).

[11] GROSS, T., P.STEENKISTE, AND SUBHLOK, J. Adaptive distributed applications on heterogeneous networks. In *8th Heterogeneous Computing Workshop* (Puerto Rico, April 1999). Invited paper.

[12] HARCHOL-BALTER, M., AND DOWNEY, A. B. Exploiting process lifetime distributions for dynamic load balancing. In *1996 ACM SIGMETRICS* (May 1996), pp. 13–24.

[13] INOUYE, J., CEN, S., PU, C., AND WALPOLE, J. System support for mobile multimedia applications. In *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video* (St. Louis, May 1997), pp. 143–154.

[14] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor — A hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems* (San Jose, California, June 1988).

[15] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource

query interface for network-aware applications. In *Seventh IEEE Symposium on High-Performance Distributed Computing* (Chicago, IL, July 1998).

[16] PAXSON, V., AND FLOYD, S. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking 3*, 3 (June 1995), 226–244.

[17] PAXSON, V., AND FLOYD, S. Why we don't know how to simulate the internet. In *Proceedings of the 1997 Winter Simulation Conference* (1997).

[18] SCHERER, A., LU, H., GROSS, T., AND ZWAENEPOEL, W. Transparent adaptive parallelism on NOWs using OpenMP. In *Proceedings of the Seventh ACM Symposium on Principles and Practice of Parallel Programming* (Atlanta, GA, May 1999).

[19] SCHOPF, J., AND BERMAN, F. Performance prediction in production environments. In *12th International Parallel Processing Symposium* (Orlando, FL, April 1998), pp. 647–653.

[20] SHARMA, S., PONNUSAMY, R., MOON, B., HWANG, Y., DAS, R., AND SALTZ, J. Run-time and compile-time support for adaptive irregular problems. In *Proceedings of Supercomputing '94* (Washington, DC, Nov 1994), pp. 97–106.

[21] STEMM, M., SESHAN, S., AND KATZ, R. Spand: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems* (Monterey, CA, June 1997).

[22] SUBHLOK, J., STEENKISTE, P., STICHNOTH, J., AND LIEU, P. Airshed pollution modeling: A case study in application development in an HPF environment. In *12th International Parallel Processing Symposium* (Orlando, FL, April 1998).

[23] SUBHLOK, J., AND VONDRAN, G. Optimal latency–throughput tradeoffs for data parallel pipelines. In *Eighth Annual ACM Symposium on Parallel Algorithms and Architectures* (Padua, Italy, June 1996), pp. 62–71.

[24] TANGMUNARUNKIT, H., AND STEENKISTE, P. Network-aware distributed computing: A case study. In *Second Workshop on Runtime Systems for Parallel Programming (RTSPP)* (Orlando, March 1998).

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

[26] WOLSKI, R., SPRING, N., AND PETERSON, C. Implementing a performance forecasting system for metacomputing: The Network Weather Service. In *Proceedings of Supercomputing '97* (San Jose, CA, Nov 1997).