

A Scalable Method for Predicting Network Performance in Heterogeneous Clusters

Dimitrios Katramatos*
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
dk3x@cs.virginia.edu

Steve J. Chapin
Department of Electrical Engineering
and Computer Science
Syracuse University
Syracuse, NY 13244
chapin@ecs.syr.edu

Abstract

An important requirement for the effective scheduling of parallel applications on large heterogeneous clusters is a current view of system resource availability. Maintaining such a view is a time consuming problem, potentially $O(N^2)$. Although CPU availability is relatively easy to monitor, interconnecting network bandwidth varies not only with network topology, but also with message size and even with respect to the load of the communicating nodes. This paper describes a method for predicting a cluster's network performance for the purpose of scheduling parallel applications. The method generates a cluster-specific network model which can predict the latency of communications between any pair of nodes in linear time and under any computational and/or communication load conditions. The paper also presents the models generated for the Centurion cluster at the University of Virginia and the Orange Grove cluster at Syracuse University. A study of the prediction accuracy of the method under various load conditions by comparison to experimental measurements indicates an average prediction error of approximately 5% with the maximum encountered prediction error of less than 9%.

1 Introduction

Effective scheduling (or mapping) of parallel applications on large heterogeneous clusters can be achieved by pursuing the best practicable match between a system's resource availability and an application's resource needs and utilization patterns. When the necessary system and application information is readily available, such a matching can be pursued with the use of a global optimization algorithm,

like simulated annealing or a genetic algorithm. Obtaining system information, application information, and finding an efficient mapping are three separate aspects of the scheduling problem. In this paper, we focus on the aspect of obtaining and maintaining a consistent current view of the resource availability of a system.

In particular, we consider two fundamental system resources per cluster node: CPU availability and network connection availability. To be useful for scheduling purposes, the resource availability information must be available on-demand and reflect the condition of the computing system at the time of the scheduling request. Obtaining such information for a large heterogeneous cluster with N nodes can be a complex problem. Although monitoring per node parameters, such as CPU availability, is relatively cheap ($O(N)$), four factors complicate the monitoring of available network bandwidth between nodes: system heterogeneity, network topology, programming environment overhead, and the combined load of computation and communication on participating nodes. System heterogeneity, network topology, and programming environment overhead are static factors, typically changing only with major upgrades of the system. With suitable profiling measurements in the absence of significant node load (as is the case after a general reboot of the "machine") we can measure the maximum system capacity both in terms of computation and communication. Programming environment overhead can have a direct effect on the communication latency. For example, our experiments using the popular MPI [1] environment on three significantly different clusters have shown a non-linear variation of latency vs. message size up to a message size of 8KB (see figure 1). Assuming linearity in this region can lead to an error of 20% or higher in the corresponding latency values.

A dynamic factor affecting communication latency is the amount of computation and communication load that is al-

*Currently with the Brookhaven National Laboratory.

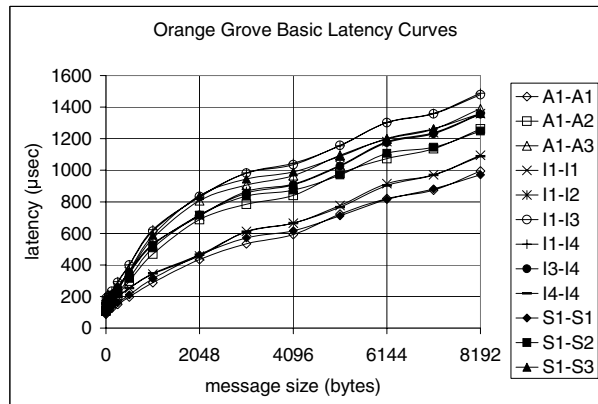


Figure 1. Non-linear region of message latency curves.

ready present in the system. This load needs to be taken into account when scheduling new tasks on the nodes of a cluster. The faster and more accurately we can measure or predict this dynamic load, the more effective our resulting schedule will be. The CPU load of a pair of nodes is easily available, but does not, alone, give an accurate predictor of dynamic communication capabilities. On the other hand, dynamically measuring pairwise communication latencies/bandwidth on a per-message-size basis provides accurate readings, but is an impractical approach because it requires $O(N^2)$ time to do a full measurement of an N -size cluster.

Our method uses the results of a sequence of microbenchmarks on a cluster to generate a model of the dynamic effect of the per-node CPU and network connection load on the latency of node pairwise communications. The model accepts two inputs:

- the CPU availability (A_{CPU}): the fraction of CPU that a newly created process will be given when assigned to run on a node, and
- the network connection availability (A_{NET}): the percentage of free bandwidth of the node's network connection

Given these quantities for a pair of nodes and a specific message size, the model can provide an estimated increase of the communication latency over the measured latency for an unloaded cluster. During runtime, the factors A_{CPU} and A_{NET} can be gathered in $O(N)$ time for the entire cluster.

Section 2 describes the background of the method while section 3 discusses the processing and correlation of the experimental profiling data and the creation of the model for a cluster. Section 4 presents a study of the method's pre-

diction error. Section 5 discusses related work and, finally, section 6 presents the conclusions.

2 Background

The computing systems of interest incorporate hundreds or even thousands of nodes and may exhibit various degrees of heterogeneity in node architecture, network architecture, network topology, operating system, etc. The two main properties of the interconnecting network we use to define the basis for categorizing such systems are the minimum bisection, measured in links or paths, or equivalently the minimum bisection bandwidth, and the use of the network (dedicated vs. public). The minimum bisection is an indication for the capacity of the network regardless of the communication patterns of the nodes. The upper bound on the minimum bisection for N nodes is $N/2$, known as "full" bisection. The lower the bisection of a system, the higher the probability of contention appearing in the interconnection network and increasing the communication latency. The use of a network imposes restrictions on the origin of the traffic in that network. A network dedicated to interconnecting the nodes of a clustered system contains only internode (internal) traffic, while a public network also contains traffic from sources external to the system, which can be unpredictable. With these two properties in mind, we consider three categories (types) of clustered systems:

Type-1 Basic heterogeneous cluster with N nodes, high minimum bisection and dedicated network.

Type-2 Federated cluster composed of type-1 clusters interconnected with one or more, low or high capacity, dedicated links.

Type-3 Federated cluster composed of type-1 and/or type-2 clusters connected with public links or the Internet.

In the environment of a type-1 cluster the only dominant factor dynamically affecting the latency of communications between any pair of cluster nodes is the load conditions at the two end nodes. A type-2 cluster maintains the dedicated network so that there is no unpredictable traffic. However, the network links connecting the type-1 building blocks may or may not be enough in number and/or capacity to keep the minimum bisection at a level where contention is absent or insignificant. Finally, a type-3 cluster is essentially a special grid configuration. Here, the links between the type-1 and/or type-2 clusters are neither dedicated nor do they necessarily have enough bandwidth for the basic assumption to be valid. Traffic from sources external to the system can pass through the public links at any instant, so the available bandwidth can vary with time in unpredictable ways.

We use the results of a set of microbenchmarks on a cluster to generate an approximate model of the end-to-end communication latency between cluster nodes. The model takes into account the dynamic effects of the per-node CPU and network connection load on the “nominal” latency (minimum latency measured under no-load conditions) of node pair-wise communications. The model is realized as:

- a 3D matrix providing the no-load communication latencies for every possible pair of nodes and for a range of message sizes, and
- a set of correction coefficient parametric curve families providing corrections to nominal latencies due to node CPU and NIC load.

The model accepts as inputs:

- the identities of the sending and the receiving node,
- the message size,
- the CPU availability A_{CPU} : the fraction of CPU that a newly created process will be given when assigned to run on a node; if N_{CPU} the number of CPUs of a node, and \overline{LOAD}_{CPU} the current average CPU load then A_{CPU} is defined as:

$$\frac{N_{CPU}}{1 + \overline{LOAD}_{CPU}} \quad (1)$$

A_{CPU} is equal to 1.00 for nodes free of load; and

- the network connection availability A_{NET} : the percentage of free bandwidth of the node’s network connection.

The output is an estimated value of the actual communication latency a message of given size will encounter when sent from the sender to the receiver node under current load conditions. During runtime, the factors A_{CPU} and A_{NET} can be gathered in $O(N)$ time for the entire cluster.

The presented model estimates properly the pair-wise communication latencies when the internode latency is primarily affected only by node loads. Type-1 clusters follow this premise by definition. Type-2 and type-3 clusters, however, are bound to encounter common link contention problems. To account for this kind of contention, we extend the basic model with an additional latency correction. In type-2 clusters, we calculate this correction by periodically sampling the latency of a shared link by running a benchmark on at least one characteristic node pair connected by that link. Sampling more than one pair increases the confidence in the results. If the measurements show consistently a higher latency value than the one estimated by the basic model, contention is present and the additional correction is

the ratio of the measured value over the basic model estimated value.

In type-3 clusters there is the additional issue of unpredictable shared link bandwidth fluctuations with time due to traffic external to the system. Because of these fluctuations, it is practically impossible to populate the nominal (no-load) latency matrix with representative values for node pair-wise communications that utilize the shared link. The extension to the model is similar to the extension for type-2 clusters, i.e. periodic sampling of the latency between characteristic node pairs to determine the condition of the link. However, the selected node pairs must stay free of load at all times because it is impossible to distinguish between latency increases due to external traffic or due to node load. By eliminating the possibility of an increase due to node load, each measurement returns the (instantaneous) latency of the link between the node pair. A second issue is the population of the region of the no-load latency matrix corresponding to the shared link. In this case, the no-load latency for a node pair can be either the minimum observed no-load latency value over a period of time, as this value is highly probable to correspond to a free link, or the theoretical minimum latency estimated from the characteristics of the hardware and the topology of the network. The extended model now applies to type-3 clusters as to type-2 ones; it estimates a correction to nominal latency due to node load and a second correction due to link contention.

3 Correlation Methodology

The model of the internode latency behavior of the cluster is generated by correlating the parameters measured during the benchmarking phase. The goal is to obtain a meaningful function that expresses the variation of latency vs. a set of parameters measured in $O(N)$ time (in this case, A_{CPU} and A_{NET}), i.e. a function of the following form:

$$\frac{L}{L_0} = \mathcal{F}(A_{CPU}, A_{NET}) \quad (2)$$

Each pair of nodes chosen to represent a characteristic group of cluster nodes requires two series of experimental data (for a predetermined set of message sizes). The first series consists of measurements of $\frac{L}{L_0}(t)$ and $A_{CPU}(t)$ under the imposition of pure CPU load and the second series measurements of $\frac{L}{L_0}(t)$, $A_{CPU}(t)$, and $A_{NET}(t)$ under the imposition of composite (network and CPU) load. Pure CPU and/or composite load is imposed artificially on the system by running in the background synthetic, computation and/or communication intensive jobs and increasingly reducing A_{CPU} and A_{NET} . Measurements for all parameters in a series are taken for N_i periodic instants t_j within an interval Δt_i , during which the load remains steady. Although desirable, it is not a requirement that the Δt_i periods

be equal. For each interval Δt_i the corresponding value of each parameter P is computed as the average of the values sampled during that period, i.e.:

$$P_i = \frac{\sum_{j=1}^{N_i} P_j}{N_i} \quad (3)$$

For each parameter, the computation of the average yields a set of i value pairs $(P, \Delta t)_i$ which are distinct values of the function $P(t)$. Because each Δt_i is common for all parameters in a series, it is possible to create a direct one-to-one correspondence between parameter values, in effect eliminating the parameter time.

Processing of the first experimental data series samples the normalized latency as a function of CPU availability:

$$\frac{L}{L_0} |_{CPU} = \mathcal{F}(A_{CPU}) \quad (4)$$

Processing of the second experimental data series samples the normalized latency as a function of CPU and network availability:

$$\frac{L}{L_0} |_{composite} = \mathcal{G}(A_{CPU}, A_{NET}) \quad (5)$$

The second series also samples CPU availability as a function of network availability:

$$A_{CPU} = \mathcal{Q}(A_{NET}) \quad (6)$$

The latter function expresses the variation of A_{CPU} vs. A_{NET} for this specific series. To isolate the network connection load effect we assume that the normalized latency corresponding to the composite load, which is a function of CPU and network availability can be expressed as the sum of a function of solely the CPU availability plus a function of solely the network availability:

$$\frac{L}{L_0} |_{composite} = \mathcal{G}_1(A_{NET}) + \mathcal{G}_2(A_{CPU}) \quad (7)$$

This assumption is based on the fact that the network connection is controlled by a separate asynchronous subsystem. Since the effect on the latency from pure CPU load is already known from the processing of the experimental data of the first data series, is possible to sample the normalized latency as a function of the network availability:

$$\frac{L}{L_0} |_{NET}(A_{NET}) = \mathcal{G}_1(A_{NET}) \quad (8)$$

This is done by eliminating the \mathcal{G}_2 portion approximated as:

$$\mathcal{F}(\mathcal{Q}(A_{NET})) \quad (9)$$

thus yielding:

$$\mathcal{G}_1(A_{NET}) = \mathcal{G}(A_{CPU}, A_{NET}) - \mathcal{F}(\mathcal{Q}(A_{NET})) \quad (10)$$

We now introduce the terms “correction for CPU load” C_{CPU} , “correction for network connection load” C_{NET} , and “total correction for load” $Corr_T$. Knowing the values of L_0 and $\frac{L}{L_0}$ directly leads to finding the value of L . $\frac{L}{L_0}$ represents $Corr_T$ and is indeed the ratio of L over L_0 under load conditions. The described process generates the desired corrections:

$$C_{CPU} = \frac{L}{L_0} |_{CPU} \quad (11)$$

the correction for CPU load, also a ratio with values ≥ 1 , and:

$$C_{NET} = \mathcal{G}_1(A_{NET}) \quad (12)$$

the correction for network connection load, now a ratio ≥ 0 . The latter correction cannot be used by itself but only in conjunction with C_{CPU} to form the total correction. This is in accordance with the fact that it is not possible to have 100% pure network load without at least a small portion of CPU load. Thus:

$$Corr_T = C_{CPU} + C_{NET} \quad (13)$$

with C_{NET} being zero when there is no network activity.

The final step in this procedure is to generate two families of correction curves for each characteristic category of cluster nodes. For the case of pairs formed with nodes of different characteristic categories, the necessary corrections can be derived from the curve families of both node categories. These families of curves give the C_{CPU} and the additional C_{NET} for any message size.

Summarizing, the full procedure to obtain an approximation of the network latency L between two cluster nodes A and B, for message size m_s , comprises the following steps:

1. Get zero-load latency for nodes A and B and message size m_s , L_{0A,B,m_s} , from the no-load latency matrix.
2. Get current CPU availability for nodes A and B, A_{CPU_A} and A_{CPU_B} .
3. Get current network connection transmitting and receiving availability for nodes A and B, $A_{NET_{A_{tx}}}$, $A_{NET_{A_{rx}}}$, $A_{NET_{B_{tx}}}$, and $A_{NET_{B_{rx}}}$.
4. Get values for CPU correction for nodes A and B, C_{CPU_A} and C_{CPU_B} , from the appropriate curves of the model corresponding to each node category.
5. Compute the correction for CPU as the following mean:
$$C_{CPU} = \frac{C_{CPU_A} + C_{CPU_B}}{2} \quad (14)$$
6. Get values for network connection correction for A and B from the appropriate curves of the model, using $C_{NET_{A_{tx}}}$, $C_{NET_{A_{rx}}}$, $C_{NET_{B_{tx}}}$, and $C_{NET_{B_{rx}}}$.

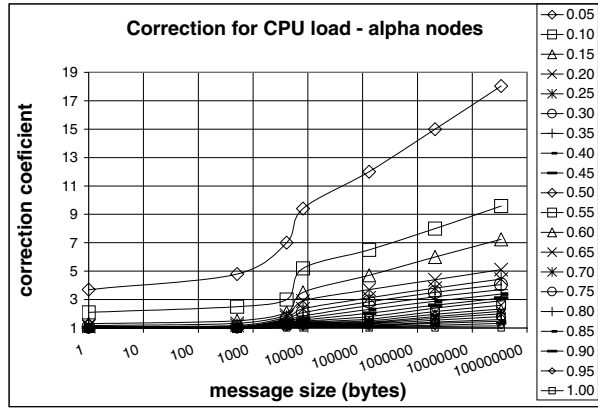


Figure 2. Correction for CPU load, Alpha nodes

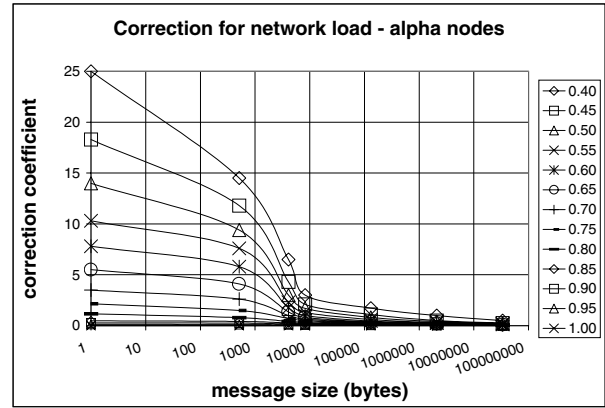


Figure 3. Correction for network connection load, Alpha nodes

7. Compute the additional network correction as the following mean:

$$C_{NET} = \frac{C_{NET_{A_{tx}}} + C_{NET_{A_{rx}}} + C_{NET_{B_{tx}}} + C_{NET_{B_{rx}}}}{4} \quad (15)$$

or

$$C_{NET} = \frac{\overline{C_{NET_A}} + \overline{C_{NET_B}}}{2} \quad (16)$$

8. Total correction is given as:

$$Corr_T = C_{CPU} + C_{NET} \quad (17)$$

9. and finally the estimated latency value is:

$$L = L_{0_{A,B,m_s}} * Corr_T \quad (18)$$

4 The Resulting Models

The Centurion cluster at the University of Virginia [2] is a 256-node cluster with 128 Intel and 128 Alpha processor-based nodes running Linux. The nodes are connected to 24-port fast Ethernet switches which are in turn connected to a 12-port 1bps central switch. Figures 2 and 3 present correction curve families for the Centurion nodes of Alpha architecture. Note that the message size axis is logarithmic to accommodate the range of useful message sizes. These curves are essentially a fingerprint of the effect of CPU and network load on the message latency for the specific hardware and software of the cluster. An immediate observation for Centurion is that the effect of network connection load diminishes with increasing message size while the opposite is true for the effect of CPU load.

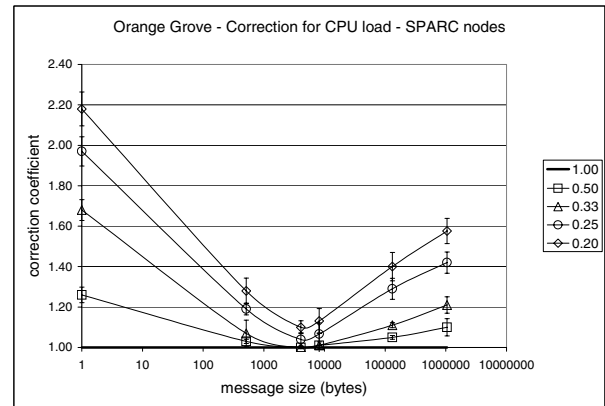


Figure 4. Orange Grove: correction for CPU load, SPARC nodes

The Orange Grove cluster at Syracuse University[†] is a 28-node cluster with 8 Alpha, 8 Sparc, and 12 Intel processor-based nodes. It is organized in 4 mixed-architecture basic blocks which in turn form two larger block pairs. There is only one network link between the two block pairs (central link) and all network connections are fast Ethernet. The cluster is configured so as to resemble a scaled down type-3 cluster. Due to the low bisection (minimal bisection bandwidth of just 100Mbps) the central link is prone to contention effects. Figure 4 gives an example of what the correction coefficient curve families look like for this cluster.

The coefficient curves of Centurion present a sharp drop of the correction coefficients for message sizes in the re-

[†]We used a re-arranged version of the original Orange Grove cluster.

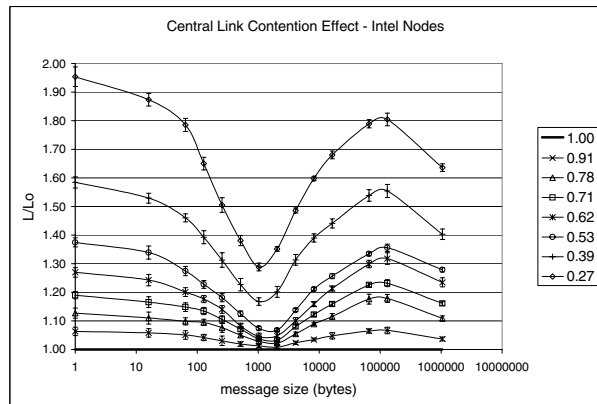


Figure 5. Central link contention effect, Intel nodes.

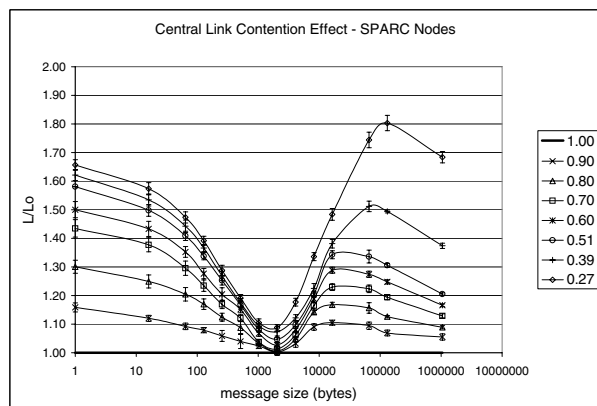


Figure 6. Central link contention effect, SPARC nodes.

gion of 1KB to 8KB, regardless of architecture. However, for increasing message sizes, the coefficient values for CPU load correction are consistently increasing, and, for network load correction, consistently decreasing. In contrast, the coefficients for Orange Grove present a minimum in the same message size region. This observation holds for all three node architectures used in the Orange Grove cluster.

Figures 5 and 6 present the contention effect at the central link as seen from the perspective of Intel and SPARC nodes, respectively. Both curve families present a minimum in the message size range of 1KB to 8KB, similarly to the curve families of figure 4. This fact indicates that if programs use message sizes in the system's "preferred" range, they will encounter minimal contention effects.

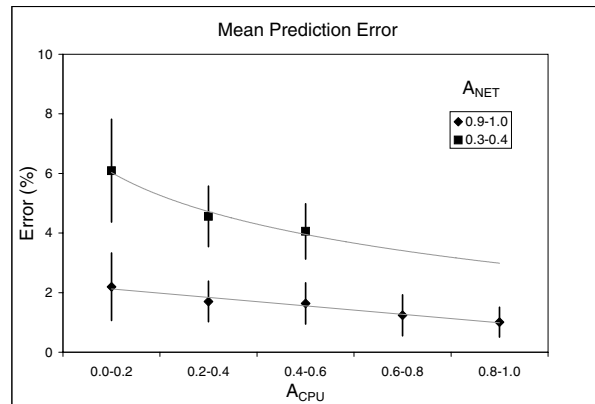


Figure 7. Prediction error.

5 Model Evaluation

To evaluate the results of the models generated for Centurion and Orange Grove, we measured the actual pair-wise latency for several cases of node pairs under different conditions of both real and artificial load. We compare these measurements to the predictions of the models Figure 7 offers a collective view of the results obtained from both clusters, with parameter sweeps for A_{CPU} , A_{NET} , and message size, and for a variety of node pairs. The node pairs sample the node combination space with regard to hardware architecture and network topology. The figure displays the mean prediction error and the corresponding 95% confidence intervals from the full range of encountered A_{CPU} values and for the upper ($A_{NET} = 0.9 - 1.0$) and lower ($A_{NET} = 0.3 - 0.4$) limits of the encountered A_{NET} values. In essence, the figure indicates the range of prediction error. We did not observe any significant dependence of the prediction error on node category mix and/or message size. The error remains virtually constant at approximately 2% for high values of A_{NET} . However, there is a strong indication that model accuracy decreases with increasing network load. This is because large loads cause larger fluctuations of the various parameter values (larger standard deviations). The maximum error encountered during the comparison is approximately 9%, but such error levels correspond to extremely loaded node pairs. For all practical purposes, the mean error is 5% or less.

6 Related Work

The performance of the message passing layer, as part of the runtime system of a cluster, is critical for message passing applications. In our approach, the use of microbenchmarks is a key element in profiling and monitoring system resources. The design of these benchmarks has

its basis in the information presented by Liu, Culler, and Yoshikawa [3]. They use microbenchmarks to evaluate MPI performance on three different hardware platforms, each with its own implementation of MPI, and reveal the extent that hardware architecture and underlying implementation of MPI affect message passing performance.

There are several research efforts dealing with the problem of monitoring resource availability in large-scale heterogeneous distributed systems. One of the most widely used works in system resource monitoring in heterogeneous distributed systems, is the Network Weather Service (NWS) [4]. NWS uses sensors (daemon processes) to monitor the resources of each host participating in a distributed system and a set of centralized administrative and database components to coordinate measurements and store results. Groups of sensors can be organized in NWS cliques, structures that enforce a conflict-free pair-wise benchmarking protocol. NWS can collect sensor data for all measured parameters periodically and create time series and has the capability of forecasting parameter values for near-future periods. To deal with the scalability problems NWS encounters when gathering $O(N^2)$ latency measurements, Swamy and Wolski [5] proposes dividing a system's nodes into categories and selecting representative nodes from each category to be members of NWS cliques, thus creating a hierarchical clique structure. Recent work by Casanova et al. [6] uses NWS traces to model CPU load and network latency/bandwidth while examining heuristics for scheduling parameter sweep applications in grid environments through simulation.

Remos [7] is a system designed to provide distributed applications with resource information and supports resource measurements in various environments and for various applications. The system incorporates collectors, modelers, and predictors. The collectors acquire network resource information and forward it to the modelers; the modelers expose the Remos API to applications and do all necessary processing to respond to queries made by applications through the API. The predictors turn a measurement history into a prediction for future behavior. The collectors use either SNMP to collect information directly from routers and switches or explicit benchmarking similarly to NWS; they can operate both on-demand and periodically and rely on aggressive information caching to reduce the $O(N^2)$ overhead.

Topology-d [8] is a service that periodically estimates end-to-end latency and available bandwidth among N networked resources (N^2 paths). Topology-d uses the estimated delay and bandwidth values to compute a fault-tolerant, minimum cost spanning tree that connects the participating sites. The service uses a "damping" effect to take previous measurement history into account and tries to measure the delay and bandwidth as actually "seen" by an ap-

plication.

Network Tomography [9] uses end-to-end measurements of a network to infer internal behavior. Internal network delays, common portions or routes between different sets of senders and receivers, and other information can be obtained without having to perform measurements at internal points of the network.

The Autopilot system [10] is an infrastructure for dynamic performance tuning of heterogeneous computational grids. Autopilot sensors, inserted in application or library code, can either capture and transmit raw data or compute and periodically transmit application or system characterization metrics. When an application executes, the embedded sensors register with a name server provided by an Autopilot Manager. Remote clients can then query the manager to locate sensors with specific properties and receive measurement information directly from these sensors. This information can be used to decision mechanisms to optimize resource management. Sensors, managers and clients can execute anywhere on a grid.

Finally, Ganglia [11] is a scalable distributed monitoring system for clusters and grids, with support for several operating systems. The system is based on a hierarchical design targeted at federations of clusters. Ganglia can monitor a variety of metrics, system supported or user defined, within clusters, and uses a tree of point-to-point connections amongst representative nodes of clusters participating in a federation, through which it can collect the partial states and form a global view of the federated system. Using a special tool, Ganglia can store and visualize historical data for time granularities ranging from minutes to years.

The major difference of our approach with regard to the mentioned works is the use of a cluster-specific network latency model, based on experimental latency measurements, to predict the performance of the network at scheduling time. The use of this model makes possible to predict network latencies in linear time without having to resort to a multitude of "expensive" node pair-wise latency measurements when latency information is needed within short time constraints.

7 Conclusions

We have presented a methodology for creating a resource availability model of a given heterogeneous cluster. The ultimate goal of this model is to generate, on demand during runtime, a valid view of the available resources without having to resort to exhaustive, time consuming measurements. Our approach uses an initial $O(N^2)$ round of offline measurements of internode latency to obtain the suitable parameter values and to generate the model, although careful analysis of the cluster architecture can ameliorate this affect by executing non-interfering microbenchmarks

concurrently. However, after this initial (off-line) phase, we can generate a valid resource view during runtime from system data gathered in $O(N)$ time, regardless of the node and network load conditions of the cluster. This enables us to produce results comparable to other methods with a smaller expenditure of resources. As such, our approach scales well when moving from clusters with tens to clusters with hundreds and/or thousands of nodes.

Following this methodology we generated models for the Centurion cluster at UVa and the Orange Grove cluster at Syracuse University. To evaluate our method, we conducted a comparison of predicted against measured latencies for a variety of test cases on both clusters. The test cases were selected so as to cover the parameter space with regard to CPU availability, network connection availability, node pair architecture mix, and message size. We encountered a maximum error of less than 9%, however, for light to medium loaded systems, the average error was 5% or less.

References

- [1] Message Passing Interface
<http://www.mpi-forum.org>
- [2] The Centurion Cluster
<http://legion.virginia.edu/centurion/Centurion.html>
- [3] Liu, L. T., Culler, D. E., and Yoshikawa, C. "Benchmarking message passing performance using MPI" In *Proceedings of the 1996 International Conference on Challenges for Parallel Processing*, Ithaca, NY USA, Vol. 1, pp. 101-110, 1996.
- [4] The Network Weather Service
<http://nws.cs.ucsb.edu/>
- [5] Swany, M., and Wolski, R. "Building Performance Topologies for Computational Grids" *Los Alamos Computer Science Institute (LACSI) Symposium*, October 2002.
- [6] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Heterogeneous Computing Workshop*, pages 349–363, 2000.
- [7] T. Dewitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland. ReMoS: A resource monitoring system for network aware applications. Technical Report CMU-CS-97-194, School of Computer Science, Carnegie Mellon University, December 1997.
- [8] K. Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. In *Proceedings of the ACM SIGMETRICS SPDT'98*, October 1997.
- [9] M. Coates and R. Nowak. Network Tomography for Internal Delay Estimation. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Salt Lake City, Utah, May 2001.
- [10] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed. Autopilot: Adaptive Control of Distributed Applications. In *Proceedings of the High-Performance Distributed Computing Conference*, July 1998.
- [11] The Ganglia Toolkit
<http://ganglia.sourceforge.net/>