Network Adaptability in Clusters and Grids

Jeffrey J. Evans
Department of Electrical and
Computer Engineering Technology
Purdue University
401 N. Grant St.
West Lafayette, IN 47907
jje@purdue.edu

Seongbok Baik, Joseph Kroculick, Cynthia S. Hood
Department of Computer Science
Illinois Institute of Technology
10 W. 31^{st} Street
Chicago, Illinois 60616
sbbaik@iit.edu, krocjoe@ieee.org, hood@iit.edu

***Abstract*— The acceleration in computational scale to solve problems in emerging "computational" fields from Nanoscience and Genetics to Astrophysics places increasingly heavy compute and data storage burdens on locally and globally distributed computer systems. We are focusing on the management of these loosely coupled systems (clusters and Grids) which are asked to behave as an increasingly large single entity, repeatably and reliably. Our approach explores several areas and levels, from low level detection and reaction of loosely coupled subsystems to application dynamics, including human factors. These areas are discussed and results from our work to date is presented. New questions emerge to stimulate deeper thought and discussion of requirements understanding and capture.**

Keywords: Cluster, Grid, Network Performance, Application Run-Time Sensitivity.

I. Introduction and Background

A central idea in high performance distributed computing is that the entire computing environment; resources, management and control entities, and the set of running applications, be thought of as a *"system"*. It is desirable that the system operate in a consistent manner. Specifically, inconsistent or highly variable application run-time is an issue, particularly on cluster systems [15], [23] composed of networks of workstations (NOWs) [1] and larger, more heterogeneous Grids [17]. Highly variable run-time carries both technological and economic ramifications.

The motivation for using large compute and storage resources is clear. There are now cases where insufficient compute and/or data storage resources are available in one location to perform a computational task in a reasonable time, if at all. Research in genetics is an example where large compute and storage resources are needed. In one study of phylogenetic inference, taxonomic groupings of organisms can require the creation of an excess of 10^{300} bifurcating trees [43]. Complete analysis of a sufficient number of randomizations (200) of this data structure would take roughly five years with one machine, but was reduced to about one month using 64 processors.

In the area of atmospheric modeling, a fine-grained domain decomposition in three dimensions can require 10^5 to 10^7 tasks [16], while in efforts to study fundamental structures of matter, a single iteration of a conjugate gradient algorithm used in quantum chromodynamics[1] simulations requires on the order of 10^9 floating point operations [12]. In the commercial sector, where the scale of computational resources still dwarfs that of the research community, web servers are increasingly adopting forms of parallelism in the form of clustering to reduce response time, thereby keeping their customer engaged. All of these problems tend to use parallel programming techniques to achieve the desired result.

These examples share the idea that the entire computing environment (the application and the resources it uses) must be thought of as one system. They also share the goal of consistent system performance. This means that the system which includes the application code, compute and I/O nodes, and the interconnection network performs consistently over time when operating conditions are similar. Performance consistency is critical so that orderly and fair resource usage is maintained.

A lack of performance consistency carries economic cost implications as well. Cluster and Grid computing is evolving from the research environment to include the business environment. As this evolution progresses, it will quickly become unacceptable for providers to deliver inconsistent performance of resources while reaping the financial benefits of extended application run-times. Moreover, it will also become unacceptable for providers not to control the performance of their resources or even explain why inconsistency is occurring relative to their resources.

The cause of highly variable run-times can be an elusive target. Competing communication patterns can result in network congestion and are often blamed for poorly behaving applications. Components of the computing environment itself are in a constant state of flux. Computing resources, their management and control software, and network components are constantly being tuned or upgraded. The set of applications is highly dynamic with their commencement and termination. Alternatively, mis-configuration of network elements required to interoperate with a plethora of protocols across several communication layers generally goes unnoticed until performance sufficiently degrades [27]. Subsystems including

[1]QCD simulations are used to predict the existence of "exotic matter" that contains gluons as an essential ingredient.

schedulers and resource managers are often left in their default operational configuration, thus not taking advantage of local environment details.

Our work focuses on the contribution of network performance to the issue of service, and subsequently application performance variability. It is widely recognized that inter-process communication in distributed systems is costly and therefore should be minimized whenever possible. Yet, the realities of parallel application program development such as reuse and/or development time force concessions in the areas of partitioning, agglomeration, and process mapping. These concessions frequently result in less than optimal inter-processor communication strategies either in volume or pattern. Moreover, traditional techniques to model communication costs generally do not consider second order effects such as contention, re-transmission, and the cost of re-routing in adaptive schemes. These effects can undermine one of the primary objectives of inter-processor communication in distributed systems, namely application synchronization.

Research has been performed and is ongoing in each of the areas just described. The vast majority of this work strives to improve performance in a single area without a clear understanding of outside influences. We can put these areas into three general categories: application, resource allocation and network. One can think of these categories as layers with the application layer on the top, the resource allocation layer in the middle and the network layer on the bottom. Using this characterization we can draw the analogy to a network protocol stack.

In the protocol stack each layer provides services to the layer above it. Each layer has capabilities related to services provided within the layer. The capabilities correct errors or adapt so the layer may continue to provide service under a variety of conditions. In addition, each layer has capabilities to compensate for the lack of adequate service from the layer below. This increases the robustness of the system. Although it is good to have redundant capabilities, it is understood that a problem can usually be best solved closest to where it occurs. An accurate description of the problem or performance degradation can be obtained and the capabilities of the layer can be used. As you move further away from the problem it becomes more difficult to pinpoint the cause of the problem and address the underlying issue. Our goal then is not to improve performance per se, but to understand those factors that most significantly influence performance consistency.

The remainder of this paper is organized as follows. In section 2 we briefly review areas of related work to reinforce our assertion that subsystem interactions are generally not considered. In section 3, we present a brief review of work done at the lowest levels of our system classification; the network. This work deals with adaptive routing techniques and the relationships of device provisioning (or lack thereof) and service survivability. Section 4 examines work related to application and system dynamics. Here we explore application run-time variability in terms of network performance and human factors relating to large compute system use. Finally, we summarize our thoughts and target areas for future work in section 5.

II. Related Work

As mentioned earlier, our work focuses on loosely coupled sub-systems. Sub-systems in this category include communication networks, schedulers, resource managers, and the applications themselves. These sub-systems are generally designed and developed in isolation of each other. Clusters we are studying generally allocate a node or set of nodes to a user, thus a single parallel application (along with each machine's operating system overhead) executes until completion. For the purposes of this work we ignore the interactions of tightly coupled resources, such as the CPU and memory [52].

A. Cluster Networks and Adaptability

One of the key issues for clusters is the interconnection mechanism. Since the nodes do not physically share memory, they rely on message passing through the network [34]. To achieve good performance in message-passing multi-computer systems, low latency as well as high bandwidth is required. In the communication network area, latency has not been a major requirement, particularly in best-effort networks. In a single-processor system, if the memory-access latency exceeds approximately one instruction time, the processor must idle until the storage cycle completes. It follows that in multi-processor systems, network latency should be small to minimize the waiting time of each computer for the response from others at each instruction cycle [2]. Existing LAN technology (i.e. Fast Ethernet, Gigabit Ethernet) can provide adequate bandwidth for cluster interconnections. The weakness with these technologies is the lack of latency bounds.

To address both the bandwidth and latency requirements of clusters, a specialized type of network called a System Area Network (SAN) [22] has been developed. SANs provide both low latency and high bandwidth [8]. Cut-through and wormhole routed technologies have emerged as the predominant base interconnection technology, used within the research community. Examples of technologies based on this principle are Myrinet [7], the Quadrics Interconnection network (QsNet) [38], and the latest version of the High Performance Parallel Interface (HiPPI), called HiPPI-6400 or also referred to as the Gigabyte System Network (GSN) [48].

There has been considerable effort put forth in achieving maximum performance of network hardware in terms of raw bandwidth. Several techniques have been developed at the sub-application level to achieve communication abstraction and latency improvement. These include special message passing libraries such as MPI [20] and PVM [18]. Additionally, techniques have been developed to all but eliminate kernel intervention of the communication process with zero copy mechanisms.

High-performance network adaptability has been studied extensively in terms of correctness [10], [19], deadlock detection [29], [26], and later traffic management [24]. Congestion

control [5], [9] including hot-spot contention [11], [40] has also been studied.

Mechanisms have been proposed to help curtail network performance degradation. Several of these even discuss the stimuli desired to make the network more adaptive. Given these stimuli, the network can react autonomously to re-route traffic to where it *believes* performance will be improved. Ideally, it would be nice to predict exactly when communications will occur and what resource capacity is required. First order modeling of parallel applications assumes the availability of full resource capacity whenever needed [16]. Second order analysis of contention generated by the application itself can result in accurate predictions of speedup. The problem gets much more complicated when multiple applications share communication resources, as is the case with most clusters and certainly Grids.

The paradoxical question then is *at what level of network performance degradation does the application lose enough synchronization to jeopardize run-time consistency?* The paradox is that the application must somehow know what this limit is (its sensitivity to communication performance). Second, the system, (characterized by applications, resource managers, and network), must somehow be made aware of, disseminate and use this information in a constructive way.

B. Scheduling and Resource Management

The function of a scheduler in a cluster or Grid environment is to allocate (map) nodes to jobs. So we say that the scope of the scheduler is system wide. For example, a cluster scheduler generally is responsible for allocating nodes within a cluster to jobs. Closely related to schedulers are resource managers whose focus is component related. That is, a resource manager is more concerned with individual components, such as a single compute node, a set of storage nodes, or a portion of an interconnection network. For instance, a resource manager typically handles activities such as moving a parallel program's application code to the nodes, then beginning the execution of the program. The problem of efficiently scheduling groups of tasks to a set of machines forming a system is *NP-complete* [13], [37].

The objectives of each are clear. For cost effectiveness, the scheduler must always try to fully utilize the nodes within the system. The objective of a resource manager is to utilize each system component effectively. An example of this might be to match compute node capability in a cluster with high performance network resources with an application that can most benefit from these resources. Interestingly, most schedulers available today work well only on certain systems and were never designed specifically for use on clusters [6].

In cluster environments, distributed techniques such as [13] propose local scheduling based on coscheduling techniques. Work related to centralized cluster scheduling include communication-aware task mapping strategies [37], contiguous allocation strategies with [23] or without aggressive backfilling [30], [44]. At this time however there is no evidence of combining the gathering of communication requirements or the converse (application run-time sensitivity to communication performance) with the integration of centralized scheduling and resource management systems.

C. Application Performance: Monitoring, Analysis and Tools

Extensive research has been performed during the last decade in the area of application level performance monitoring and analysis. Several approaches to application level monitoring have been proposed including entity-relationship (E-R) modeling [35] agent-based [42] and other on-the-fly techniques for selection of what to monitor when [32]. This was in recognition that collecting large amounts of performance data contributed to application perturbations, increasing application instability.

Performance analysis is a natural extension to the monitoring efforts above. Several research projects are in progress developing techniques and tools to address this area. These tools primarily use application and kernel code instrumentation to provide performance measurement (monitoring) and subsequent analysis services [32], [35], [39], [42]. The main objectives for these tools are to use software means to better monitor, analyze, diagnose, and then tune parallel applications, without unduly perturbing application program execution. It is unclear in these bodies of work whether, or to what degree the communications network is viewed as an integral component of the overall application, like other hard devices such as a disk system.

On the scale of metacomputing (Grids), network prediction for the purposes of dynamic scheduling is considered using the Network Weather Service (NWS) [50]. In this case communication experiments are conducted between hosts to measure network health. This data is then used to make predictions about future network health. Predictions are mean, median, or auto-regressive based. These predictions are then disseminated to all interested clients.

Program tuning [46] and steering [21], [49] is an area that shows promise toward adapting parallel applications to run more efficiently (maximizing performance) during program execution. A limitation as noted by the authors is that some programs are more steerable than others. Before a program can be tuned or steered it is necessary for the parallel program to communicate "what" can be tuned or steered, which is considered in [25]. As part of the Active Harmony [47] software architecture for computational object management in dynamic environments, applications are allowed to advertise tuning options to a higher level decision making entity.

III. Network Level Detection and Reaction

A. AM3 - Active Myrinet Mapper with Monitor

Parallel applications running on NOWs do so for a single reason: to minimize (speed-up) application run-time. As stated earlier, data movement (communications) is an expensive operation. Therefore it is desireable to take advantage of technologies that minimize both raw transmit time (seconds per bit, the inverse of bandwidth) and other latency contributors such as kernel interactions. Myrinet is one of several technologies

that uses source (wormhole) routing to virtually eliminate store and forward delays in the network [34]. This comes with the cost of moving the routing function from the network to the edges (hosts). It is intuitive then to imagine the growth function in time and space (memory) for each host to calculate and store alternative, disjoint routes as the size of the cluster grows.

Myrinet uses centralized Mapper software for routing. The first step in calculating the routes is to discover the network topology. The Mapper program creates a map file and updates it by checking the status of each host Network Interface Card (NIC) on a regular basis. The Mapper monitors every host to ensure the reachability. Each host NIC runs a Mapper client called *mappee* to monitor NIC status. To maintain high availability, it is critical that faults or performance problems are detected and the routes are updated in a timely fashion. The Mapper has an active mode for dynamic routing.

Unfortunately, the existing Mapper software has displayed documented problems when running in active mode [51]. The result is that this mode is not used, significantly reducing the fault tolerance. In addition, the Mapper might not reflect the correct status of each host because the mappee is running on the network interface card (NIC) separately. In some harsh conditions, the NIC can be working even though the host machine is not functioning properly. Other versions of the Mapper also rely on clients running on the NIC [31].

We have designed a new mapper: AM3 (Active Myrinet Mapper with Monitor) [4]. AM3 distributes the Mapper functionality and has host-based clients (mappees). The goal of AM3 is to:

- improve the memory efficiency of the existing Mapper.
- improve alternative route calculation in terms of fairness and the disjointness of routes.
- extend the monitoring capabilities to the hosts.

AM3 distributes the routing traffic by creating a hierarchy of mappers. It also takes advantage of regularity in terms of switch connectivity. Myrinet probing messages are emitted in greedy fashion potentially causing deadlocks in the worst case [33]. AM3 is significantly more efficient than the Myrinet mapper in terms of probe messages.

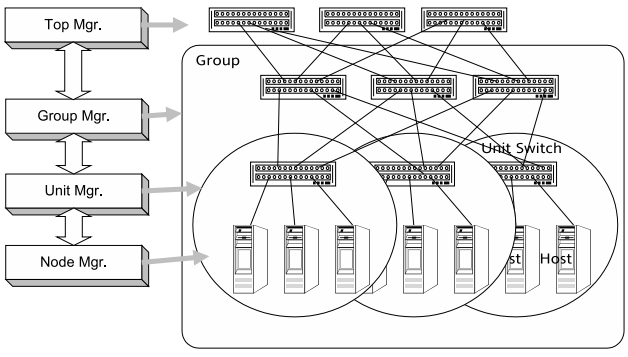

Fig. 1. AM3 Hierarchy [3]

Preliminary results include the recent development of an adaptive source routing algorithm that takes advantage of Clos Network characteristics. Myrinet products follow the Clos Network characteristics faithfully and keep the connection regularity [41]. The assumption is that the connectivity is very regular because it meets the Clos network characteristics. This algorithm reduces the number of probing messages significantly and it also uses its knowledge of the topology to remove the deadlock hazard. The calculation of the alternative routes becomes simple and speedy, because the scheme uses different hierarchical equations to produce different routes.

In [3], experimental results comparing memory cost and route calculation characteristics report a nearly 80% reduction in host memory cost while producing even distribution of routes and fairness superior to Dykstra's algorithm and comparable to the vendor's ad hoc algorithm.

B. Network Element Provisioning and Survivability

Extending the idea of network survivability and adaptability is crucial to wide-area (global) high performance computing. This area is commonly known as *Grid* computing [17] and has been rapidly gaining attention. We speculate that the area of run-time performance management in these environments will grow in importance as the area of grid computing evolves. Clearly a user who is paying for the use of widely distributed computational resources will insist on consistent run-time performance. Additionally, suppliers of these resources will ultimately find it difficult to efficiently schedule these resources and further run the risk of lost revenue due to balking or reneging.

It is necessary then, to better understand performance degradation drivers by studying performance relationships in both the horizontal and vertical planes. By horizontal we mean those causal relationships that occur between 'peers', at any layer (physical, link, kernel, application, etc.). By vertical we mean those causal relationships between layers, adjacent or not, all the way up through the operating system and into the applications and central control entities.

In the lower levels of the horizontal plane, integrated recovery solutions become even more critical due to an increasing number of *client* layers (such as IP and MPLS) expecting services from *server* layers like SONET or even WDM. The high performance computing area places enormous requirements not only on computational resources, but also data storage and movement.

Performing large scale computations over global networks requires a high degree of interoperability and provisioning consistency among the network elements. One question that arises is that of survivability requirements for high performance Grid applications. In the case of a failure somewhere in the global network, should all traffic be restored in bulk, using traditional techniques such as those used to provide SONET protection?

There may indeed be sufficient justification to employ recovery at different layers. In this case, the coordination of recovery at the different layers becomes more critical. Separate recovery mechanisms at the different layers can occur

simultaneously, increasing the possibility of uncoordinated recovery.

Another area we are investigating is that of consistency checking of provisioning in multilayer networks. The goal is to specify criteria and constraints on configuring multiple recovery processes. In [27], an algorithm is defined that creates an initial network topology based on network element (NE) models and key components, such as paths through a NE and interfaces of a NE.

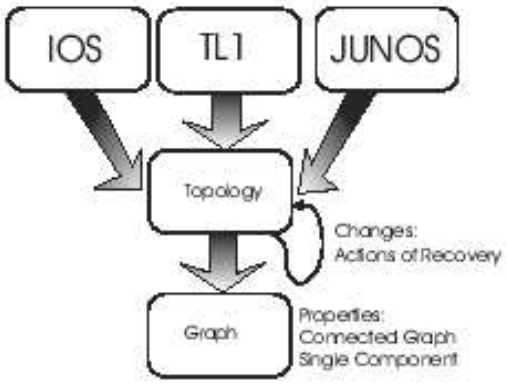

Fig. 2. CLI Command Transformation to Topological Model [27]

The creation of an initial topology is realized by way of provisioning commands issued to the NE, generally through a user interface. These commands can be highly variable, based on the NE vendor's interpretation of standards and their implementation. Once an initial topology (graph) has been created, changes to it are manifested by protection-switching commands. These commands can then be transformed into operations on the graph, such as the insertion or deletion of vertices and edges. This process is shown pictorially in figure 2. By determining the connected components using standard graph theoretic techniques, technology specific provisioning commands can be transformed into operations on the graph. These operations can then be verified against required invariants.

IV. Application and System Level Dynamics

A. Application Sensitivity to Network Performance

As stated earlier, we can think of applications, resource allocation, and network like we do a network protocol stack. Most research however tends to isolate these areas without consideration of the others. From the perspective of the application, performance tuning and "steering" concepts have been attempted to aid in post-mortem application development and optimized (tuned) execution. Research in centralized schedulers has attempted to address run-time variability in a proactive manner by trying to optimize the location of nodes while maintaining the scheduler's overall objective of maximum node utilization. Work related to network performance has traditionally focused on raw bandwidth performance, reachability, and adaptability. We are looking at this issue at the "communication" level from the perspectives of the network and application.

Our preliminary investigation of run-time variability is the analysis of run-time logs produced by the scheduler/resource management system (Maui/PBS) [45],[36] of the Chiba City [28] Linux NOW cluster at Argonne National Laboratory. The Maui scheduler has gained wide acceptance as one of the most advanced schedulers currently available to the High Performance Computing (HPC) community. Maui is what is known as a batch scheduler, which determines when and where submitted jobs should run in a sequential fashion [23].

We analyze four distinct cases previously cited in [15] where the application run-times varied significantly (more than 2x). Specifically, we are trying to establish the nature of the cluster's operating environment during application execution. Some of the questions we are trying to answer are:

- How heavily was the cluster loaded (in terms of nodes used) during the period of time where application run-time variability was observed?
- Where (topologically) were nodes allocated?

In some cases the allocation of nodes varied significantly for successive runs of the same application. To probe deeper into the cluster's behavior we investigated the operational "state" of the cluster during the time when the applications of interest were executing.

We define four distinct timeframes of interest surrounding and including a "window" of time where our applications of interest run. If we let the set of all applications that run on the cluster over a log file interval (one day) be A and the application *"set"* of interest can be denoted as A_s, then there exists a starting time $t_{s_{A_s}}$, denoting when the first instance of the application set of interest begins. There also must exist a finishing time $t_{f_{A_s}}$, which denotes when the last instance of the application set of interest finishes running. We can now define one closed and three generalized time intervals where we are most concerned with the state S_c of the cluster:

1) The closed time between and including $t_{s_{A_s}}$ and $t_{f_{A_s}}$.
2) The interval where applications were started (t_s) prior to $t_{s_{A_s}}$ and ended (t_f) after $t_{s_{A_s}}$ but prior to $t_{f_{A_s}}$.
3) The time interval where applications were started (t_s)between $t_{s_{A_s}}$ and $t_{f_{A_s}}$ but ended (t_f) after $t_{f_{A_s}}$, and
4) The time interval where applications started (t_s) before $t_{s_{A_s}}$ and finished (t_f) after $t_{f_{A_s}}$, executing *through* the time window of interest.

Pictorially this looks like:

We are then interested in the state of the cluster S_c over some application set of interest period of time where

$$S_c(t) \supseteq \begin{cases} A_s & : \quad t_{s_{A_s}} \leq t_s \leq t_f \leq t_{f_{A_s}} \\ A & : \quad t_s \leq t_{s_{A_s}} \leq t_f \leq t_{f_{A_s}} \\ A & : \quad t_{s_{A_s}} \leq t_s \leq t_{f_{A_s}} \leq t_f \\ A & : \quad t_s \leq t_{s_{A_s}} \leq t_{f_{A_s}} \leq t_f \end{cases} \tag{1}$$

The Maui scheduler log files provide both the start (t_s) and end (t_f) times of each job. We can therefore use standard

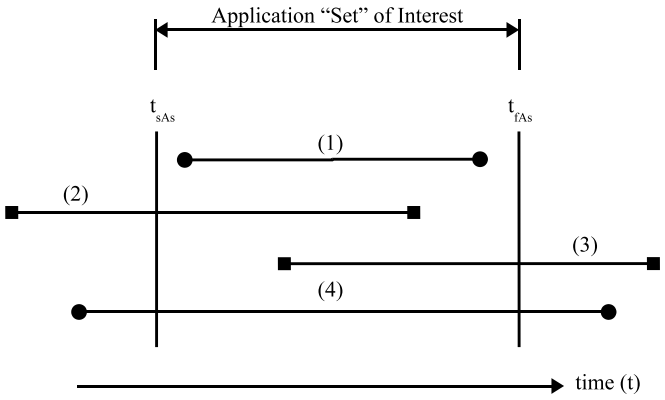

Fig. 3. Application Set of Interest Time Window

database query mechanisms and tools to expose the operational state of the cluster by joining the four conditions of equation 1.

TABLE I
RUN-TIME EVENT SEQUENCES

User	Req. (sec.)	Actual (sec.)	Req. Nodes	Node Allocation
A	28000	416	12	111 - 100
A	28000	207	12	111 - 100
A	28000	2016	12	152, 151, 149, 148, 116, 115, 113-108
B	18000	1426	72	224-218, 216, 210-201, 154, 151, 130, 119, 118, 116, 115, 113-105, 103-98, 96, 95, 93-85, 81-74, 72, 71, 69, 67-65, 32, 31, 29-27, 25, 23
B	18000	131	72	223-218, 216, 210-207, 205-201, 154, 151, 130, 119, 118, 116, 115, 113-105, 103-101, 98, 96, 95, 93-85, 81, 80, 78, 77, 75, 74, 72, 71, 66, 65, 32, 31, 29-27, 25, 23, 20, 19, 16, 13, 12, 10, 7-4
C	21600	511	32	224 - 222, 210, 207, 203, 154, 151, 130, 119, 118, 116, 115, 113 - 105, 103 - 98, 96, 95, 93, 92
C	21600	187	32	224 - 218, 216, 215, 213 - 201, 154, 151, 130, 119, 118, 116, 115, 113 - 111
D	3600	1184	6	167 - 164, 162, 161
D	3600	431	6	183 - 178

We wish to ascertain:

- the average and peak percentage of the cluster being used
- obvious congestion scenarios
- unexpected behaviors - scheduler re-mapping nodes to jobs when it might not otherwise need to.

B. Case Study: 12 Node Application

Referring to table I we note that application executions of User A produced the most widely variable run-times. We also note that User A requested 12 nodes for each of three runs. We make the simplifying assumption that the submitted parallel application problem size was equivalent for each run, since the requested job time remained constant (28,000 seconds). In the first two runs the node distribution was in reverse node-name order, from node 111 to node 100. The third run exhibited both a different node distribution and a significantly longer (10 times) run time.

We then generated a database query from the parsed Maui log file. The database query follows equation 1 in order to view the time window corresponding to the application set of interest. In this case it is the set of three runs requested by User A as well as all applications running during User A's time window. Figure 4 illustrates the results in a form consistent with figure 3. Here we use the form JobID (number of nodes allocated) to identify jobs. User A jobs are highlighted with an asterisk "*". In all cases, the jobs are presented in the order that the Maui log presented the data to the log file. We assume that the time stamps related to each job are accurate. We notice several interesting artifacts from figure 4.

First, we see that for the first two User A runs there is almost no extra activity (jobs coming and going) taking place on the cluster. Moreover, we see that those jobs that are coming and going are using few nodes (2) and are running for a very short time. These are most likely tests that failed.

Second, we now focus our attention on the last run of the set (JobID 35248). We see that at the time job 35248 begins there are 81 nodes already allocated, which is similar to the number of nodes allocated (89) when the first job (35241) was started and (81) when the second job (35243) began. However we notice that during the run time of JobID 35248 several other jobs were also submitted and started. In fact, we see that JobID's 35249 (16 nodes), 35250 (15 nodes), 35251 (20 nodes), and 35252 (2 nodes) were all started while our job 35248 (12 nodes) was running. Summing these together we have 65 nodes (including ours) added to the 81 already active nodes. This is a jump of active nodes from approximately 43% to almost 77%. Moreover, during the run of job 35248 only 15 nodes were made idle due to job termination. This represents a reduction of slightly less than 8% (from nearly 77% to just under 69%).

Upon closer inspection of the query result (and also the actual log file) we see an indication of scheduler behavior that remains unaccounted for. Specifically, we see that the node allocation for User A's third run is disbursed around the cluster (i.e. not in sequential reverse node-name order). We examine the node allocations for the jobs closest in time to 35248 and we see that job 35249 was allocated nodes 107-100, 8 nodes that had been previously allocated to our jobs of interest for the first two runs. It appears from the data however that resources for job 35249 were allocated *after* those for our job (35248).

C. Human Factors

Inconsistent performance such as that observed in the previous section is the type of system behavior we wish to better understand. Once understood, then perhaps suitable triggers and actions can be employed to better control the overall computing system.

Users on the other hand, must somehow deal with this system behavior. As reported in [15], [23], users often overestimate the run time of their program by greater than 5 times. When the program behaves as it should (terminating

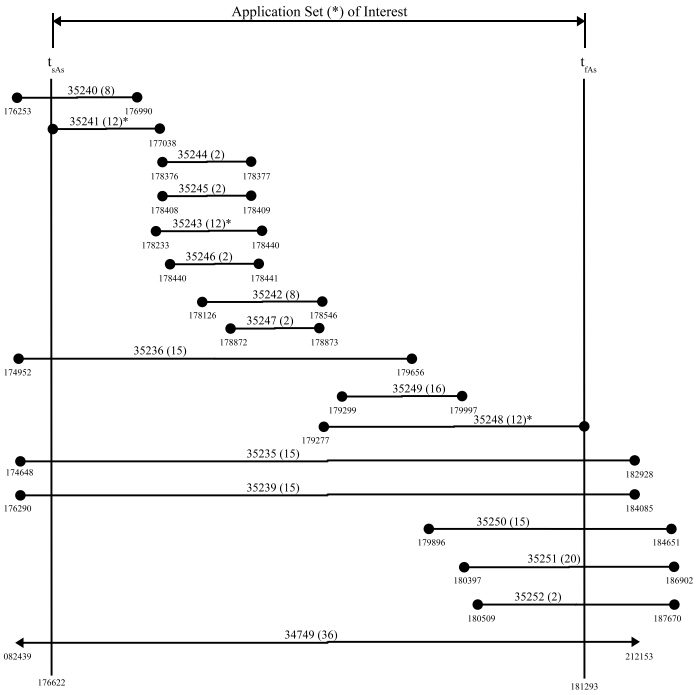

Fig. 4. 12 Node Application Set of Interest Time Window

well before the scheduler expects), the scheduler must work harder to "backfill" the just released resources with waiting jobs. This process generally requires the scheduler to re-run its algorithm, in this case at an unexpected time.

In the case of NOW clusters, the scheduler generally estimates when a newly submitted job will run. When the cluster is full of overestimated jobs, this time in the queue may be excessive, causing the user to "reneg", or delete the job from the waiting queue. this action is also an "event" for the scheduler, such that the scheduling algorithm must again be re-run.

V. Summary and Future Work

We are conducting research in several areas and on many levels to better understand the relationship between network performance and high performance parallel application run-time sensitivity. We classify the major subsystems in terms of a network protocol stack; namely applications (at the top), resource managers, and the interconnection network (at the bottom). In the network stack paradigm the lower layers provide services to the layer above it. In this case we believe it is necessary for all layers to interact with the other layers in a more proactive fashion. We plan on investigating other potential relationships between these subsystems with the purpose of combining them in a comprehensive management framework.

Parallel applications using high performance networks do so because maximum performance is expected, and therefore assumed. The interconnection network tries to move data as fast and efficiently as possible from point A to point B. In the presence of congestion, the network could autonomously decide to re-route traffic to reduce or avoid congestion, but is this the "correct" decision to make? The network must have some concept of application service expectations to effectively adapt.

Schedulers and other resource managers allocate their resources based on utilization criteria, not necessarily consistent performance criteria. A scheduler's job is to utilize as many resources as possible given the states of it's running and waiting queues, regardless of the potential ramifications due to the ever-changing "virtual" topology.

Our approach to addressing parallel application run-time sensitivity is rooted in the area of fault management. The fault management paradigm is based on monitoring a system to verify it is performing to some performance objective or requirement. When a situation arises where this is compromised (or will shortly be compromised) a corrective action is undertaken for the benefit of the operation of the system.

Our future work promises to further enhance the work presented here and further expose the network contribution to parallel application run-time variability. Once exposed, several future directions are possible. Application sensitivity "advice" could be passed to schedulers and resource managers that are equipped to use the information effectively. New models of communication cost could emerge for the benefit of parallel application developers. One of our current developments is an application communication emulation system whose goal is to characterize a parallel application's sensitivity to network performance [14]. Subsystems acting autonomously to correct perceived performance degradation may be in fact contributing to systemic performance degradation. It is therefore important to recognize that a comprehensive approach should be considered that addresses the objectives and requirements of each critical subsystem (our classification "stack") in a collective manner to achieve more consistent performance.

VI. Acknowledgements

This work was supported in part by the U.S. Department of Energy, under Contract W-31-109-Eng-38, NSF 9984811 and NSF 0325378. We are also grateful to the reviewers for their insightful and constructive comments.

References

[1] T. Anderson, D. Culler, and D. Patterson. A Case for Networks of Workstations: NOW. In *IEEE Micro*, pages 54–64. IEEE, Feb. 1995.

[2] W. Athas and C. Seitz. Multicomputers: Message-passing concurrent computers. In *Computer*, volume 21, pages 9–24. IEEE, August 1988.

[3] S. Baik and C. Hood. Decentralized route generation method for adaptive source routing in system area networks. In *The 8th World Multi-Conference on Systemics, Cybernetics and Informatics (To appear*, July 2004.

[4] S. Baik, C. Hood, and W. Gropp. Prototype of AM3: Active Mapper and Monitoring module for the Myrinet environment. In *Proceedings of the HSLN Workshop*, Nov. 2002.

[5] E. Baydal, P. Lopez, and J. Duato. A congestion control mechanism for wormhole networks. In *Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing*, pages 19–26, 2001.

[6] B. Bode, D. Halstead, R. Kendall, and Z. Lei. The Portable Batch Scheduler and the Maui Scheduler on linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, October 2000.

[7] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: a gigabit-per-second local area network. In *IEEE Micro*, volume 15, pages 29–36. IEEE, Feb. 1995.

[8] A. Chien, M. Hill, and S. Mukherjee. Design challenges for high performance network interfaces. In *Computer*, volume 31, pages 42–44. IEEE, November 1998.

[9] S. Coll, J. Flich, M. Malumbres, P. Lopez, J. Duato, and F. Mora. A first implementation of in-transit buffers on myrinet gm software. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pages 1640–1647, 2001.

[10] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. In *IEEE Transactions on Computers*, volume C-36, pages 547–553. IEEE, May 1987.

[11] S. P. Dandamudi. Reducing hot-spot contention in shared-memory multiprocessor systems. *IEEE Concurency*, 7(1), Jan.-Mar. 1999.

[12] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *Sourcebook of Parallel Computing*. Morgan Kaufman, 2003.

[13] X. Du and X. Zhang. Coordinating parallel processes on networks of workstations. *Journal of Parallel and Distributed Computing*, 46:125–135, 1997.

[14] J. J. Evans. Modeling parallel application sensitivity to network performance. Technical Report TR-NSL-03-03, Illinois Institute of Technology, May 2003.

[15] J. J. Evans, C. S. Hood, and W. D. Gropp. Exploring the relationship between parallel application run-time variability and network performance in clusters. In *Workshop on High Speed Local Networks (HSLN) from the Proceedings of the 28th IEEE Conference on Local Computer Networks (LCN)*, pages 538–547, October 2003.

[16] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Publishing Company, 1995.

[17] I. Foster and K. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[18] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, 1994.

[19] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 278–287, 1992.

[20] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 2nd edition, 1999.

[21] W. Gu, G. Eisenhauer, and K. Schwan. Falcon: On-line monitoring and steering of parallel programs. In *Ninth International Conference on Parallel and Distributed Computing and Systems (PDCS'97)*, Oct. 1997.

[22] R. Horst. Tnet: A reliable system area network. In *Micro*, volume 15, pages 37–45. IEEE, February 1995.

[23] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the $m$aui scheduler. In *7th Workshop on Job Scheduling Strategies for Parallel Processing*. SIGMETRICS 2001, ACM, June 2001.

[24] M. Jurczyk. Traffic control in wormhole-routing multistage interconnection networks. In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*, volume 1, pages 157–162, 2000.

[25] P. J. Keleher, J. K. Hollingsworth, and D. Perkovic. Exposing application alternatives. In *International Conference on Distributed Computing Systems*, pages 384–392, 1999.

[26] A. Khonsari, H. Sarbazi-Azad, and M. Ould-Khaoua. Analysis of timeout-based adaptive wormhole routing. In *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, volume 1, pages 275–282, 2001.

[27] J. Kroculick and C. Hood. Avoiding blocking in multilayer recovery. In *IEEE International Conference on Communications (To appear)*, June 2004.

[28] A. N. Laboratory. Chiba City, the Argonne scalable cluster. Online Document, 1999. http://www-unix.mcs.anl.gov/chiba/.

[29] P. Lopez, J. Martinez, and J. Duato. A very efficient distributed deadlock detection mechanism for wormhole networks. In *Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, pages 57–66, February 1998.

[30] J. Mache, V. Lo, and S. Garg. Job scheduling that minimizes network contention due to both communication and I/O. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium, IPDPS'00*, pages 457–463, May 2000.

[31] A. M. Mainwaring, B. N. Chun, S. Schleimer, and D. S. Wilkerson. System area network mapping. In *Proceedings of the ninth annual ACM Symposium on Parallel Algorithms and Architectures*, pages 116–126, June 1997.

[32] B. Miller, M. Callaghan, J. Cargille, J. Hollinsworth, B. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tools. In *IEEE Computer*, volume 28, pages 37–46. IEEE, November 1995.

[33] I. Myricom. What is the gm Mapper and how can I learn more about it?, February 2003.

[34] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb. 1993.

[35] D. M. Ogle, K. Schwan, and R. Snodgrass. Application-dependent dynamic monitoring of distributed and parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):762–778, July 1993.

[36] OpenPBS.org. The Portable Batch System. Online Doument, 1998. http://www.OpenPBS.org/.

[37] J. M. Orduna, F. Silla, and J. Duato. A new task mapping technique for communication-aware scheduling strategies. In *International Conference on Parallel Processing Workshops*, pages 349–354, 2001.

[38] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The quadrics network: high-performance clustering technology. In *IEEE Micro*, volume 22, pages 46–5. IEEE, Jan.-Feb. 2002.

[39] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera. Scalable performance analysis: The pablo performance analysis environment. In *Proceedings of the IEEE Computer Society Scalable Parallel Libraries Conference*, October 1993.

[40] H. Sarbazi-Azad, M. Ould-Khaoua, and L. M. Mackenzie. An analytical model of fully-adaptive wormhole-routed k-ary n-cubes in the presence of hot spot traffic. In *Proceedings of the 14th IEEE International conference on Parallel Processing, IPDPS 2000*, pages 605–610, May 2000.

[41] C. Seitz. Recent advances in cluster networks. Presented at the IEEE Cluster 2001 Conference, October 2001.

[42] T. Sheehan, A. Maloney, and S. Shende. A runtime monitoring framework for the tau profiling system. In *Proceedings of the Third International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'99)*, December 1999.

[43] C. Stewart, D. Hart, D. Berry, G. Olsen, E. Wernert, and W. Fisher. Parallel Implementation and Performance of FastDNAml. In *Proceedings of SC2001*, November 2001.

[44] V. Subramani, R. Kettimuthu, S. Srinivasan, and J. Johnston. Selective buddy allocation for scheduling parallel jobs on clusters. In *Proceedings of the International Conference on Cluster Computing*, pages 107–116, September 2002.

[45] Supercluster.org. The Maui Scheduler. Online Document, 2000. http://www.supercluster.org/maui/.

[46] A. Tamches and B. P. Miller. Using dynamic kernel instrumentation for kernal and application tuning. *International Journal of High-Performance Computing Applcations*, 13(3):263–276, Fall 1999.

[47] C. Tapus, I.-H. Chung, and J. K. Hollingsworth. Active harmony: Towards automated performance tuning. In *Proceedings from the Conference on High Performance Networking and Computing*, 2002.

[48] D. Tolmie, T. Boorman, A. DuBois, D. DuBois, W. Feng, and I. Philp. "from hippi-800 to hippi-6400: A changing of the guard and gateway to the future". In *Proceedings from the 6th International Conference on Parallel Interconnects (PI '99)*, pages 194–201, 1999. Proceedings from the 6th International Conference on Parallel Interconnects (PI '99).

[49] J. S. Vetter and D. A. Reed. Real-time performance monitoring, adaptive control, and interactive steering of computational grids. *International Journal of High Performance Computing Applications*, 14(4):357–366, Winter 2000.

[50] R. Wolski. Forecasting network performance to support dynamic scheduling using the Network Weather Service. In *Proceedings of the Sixth IEEE Symposium on High Performance Distributed Computing*, pages 316–325, 1997.

[51] P. Wyckoff. GM static mapping. Online Document, 2001. http://www.osc.edu/~pw/gm.

[52] L. Xiao, S. Chen, and X. Zhang. Dynamic cluster resource allocations fo jobs with known and unknown memory demands. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):223–240, 2002.