Achieving Scalable Cluster System Analysis and Management with a Gossip-based Network Service

D.E. Collins[a], A.D. George[b], and R.A. Quander[a]
[a] *HCS Research Lab, ECE Dept., FAMU-FSU College of Engineering, Tallahassee, FL*
[b] *HCS Research Lab, ECE Dept., University of Florida, Gainesville, FL*
collins@hcs.fsu.edu, george@hcs.ufl.edu, quander@hcs.fsu.edu

Abstract

Clusters of workstations are increasingly used for applications requiring high levels of both performance and reliability. Certain fundamental services are highly desirable to achieve these twin goals of network-based cluster system analysis and management. Among these services is the ability to detect network and node failures and the capability to efficiently determine computer and network load levels. Furthermore, the ability to allow for the distribution of administrative directives is also integral to the goal of cluster management. This paper presents a scalable approach to providing these vital support capabilities for distributed computing integrated into a cluster management system. Previous approaches to cluster management have suffered from problems of scalability and the inability to properly support heterogeneous systems in a non-proprietary fashion. This cluster management system employs gossip techniques to address the problem of scalability in network-based system management. The results of two case studies show that the cluster management system is scalable and has little adverse impact on the performance of sequential and parallel applications running on the managed system.

1. Introduction

The increasing shift towards distributed computing systems has accentuated the need for more scalable management and analysis tools for network-based systems. As the number of components in such a system increases, so too does the failure rate of the system and the need for effective management. In addition, the capability to efficiently measure the system loads of the nodes in the cluster is vital to achieving optimal levels of aggregate performance. Furthermore, the ability of applications running on network-based systems to tolerate network and processing node failures is intimately tied to the system management capability to detect said failures [1]. In the past, cluster management tools typically dealt with the problem of failure detection through the use of a centralized "heartbeat" protocol, wherein all nodes in the system periodically sent a heartbeat to the master node of the system. Failure to receive such a heartbeat would tell the master node that either the node in question or its interconnect had experienced a failure. Such schemes work well for small systems, but do not scale well and often possess an inherent single point of failure.

Multilevel gossiping has a number of advantages as an approach to the detection of network and node failures for a cluster management system. First, it lacks a single point of failure, as the system is consensus-based and operates in an evenly distributed fashion. It thus fails only when a majority of its managed nodes are unable to communicate with one another. In addition, when implemented as a multilevel system, gossiping has been shown to be highly scalable, both in terms of CPU utilization on each managed node and in the network bandwidth required by the service at each node.

The Cluster Management System (CMS) developed using this approach combines a gossip-based approach to failure detection with facility for the remote execution of tasks and the measurement of the system load at each node under its supervision. The remote execution and load sampling features of the CMS are similarly implemented in a decentralized manner having no intrinsic single point of failure or network traffic "hot spot." The fusion of these features gives the CMS the capability to provide information that can be used to enhance performance as well as reliability. For instance, a parallel application can identify not simply a set of nodes that are operational for its tasks, but one that is lightly loaded as well. Furthermore, the inclusion of remote task execution facilities allows management directives to be efficiently and conveniently distributed according to the desires of the administrator.

The remainder of this paper is organized as follows. Section 2 presents related research in the area of cluster management for network-based systems. Section 3 describes the characteristics of the cluster management service with a description of the design decisions made in creating the implementation. The experiments devised to test the performance impact caused by the overhead of the CMS are discussed in Section 4. Section 5 examines the results obtained from this series of experiments. The

level of overhead imposed by this design from the perspective of several representative parallel benchmarks is analyzed and compared over a range of conditions. Finally, conclusions and possible directions for future research are presented in Section 6.

2. Related research

One key area of research supporting the use of clusters of workstations connected using high-speed networks is that of the Single System Image (SSI). Using an SSI tool, a varied collection of resources can be viewed, used, and administrated as if it were a single powerful machine. SSI tools, such as Microsoft's "*Wolfpack,*" IBM's *Phoenix*, Berkley's *GLUNix* and Sun's *Solaris MC*, serve to hide much of the complexity of a cluster system and to provide many desirable services to the users and administrators [2].

Cluster management falls within the general concept of network management. Network management can be defined as the planning, organization, monitoring, accounting, and control of activities and resources within a network environment [3]. Existing tools with some cluster management capabilities, such as Scali's *ScaConf* [4], *PARMON* [5], and the various proprietary vendor tools of IBM, Microsoft, and Sun Microsystems suffer from one or more key limitations. These limitations include lack of support for heterogeneous systems, proprietary software, or design decisions in providing key services that do not scale well to very large clusters.

One such desirable service is failure detection with consensus, and it is critical to higher-level services within a fault-tolerant computing environment. The failure detection mechanisms of existing SSI environments are typically based on centralized group communications. While this method works well for small systems, it is not scalable to truly vast clusters, such as *Cplant* at Sandia National Labs. Centralized group communications also generally suffer from a lack of resiliency, as they are highly dependent on certain nodes of the system. Such an approach is often referred to as a "heartbeat" protocol. Furthermore, existing tools are generally either proprietary or intended for homogeneous systems. To meet the needs of heterogeneous clusters in a scalable fashion, a different approach is required.

Gossiping provides a scalable method of distributing information for parallel and distributed systems. Van Renesse, et al., describe the first usage of gossip techniques for failure detection in [6]. Burnes, et al., explore the performance trade-offs of various gossip protocols using simulative analysis of a Myrinet-connected cluster testbed [7]. Two types of gossiping are described in the literature: *random* and *deterministic*. In random gossiping, each node communicates with another random node each T_{gossip} seconds, whereas in deterministic gossiping, the nodes communicate in a predefined communication pattern. Ranganathan, et al., describe the first use of a failure detection system based on deterministic gossiping and demonstrate that patterned communication enables greater efficiency in [8]. They also describe and evaluate the distributed consensus algorithm chosen and adopted for the CMS.

Another gossip parameter, $T_{cleanup}$, defines the amount of time before a node is suspected as having failed after it is out of communication. $T_{cleanup}$ is a multiple of T_{gossip} and its range of permissible values is determined by the number of nodes. $T_{consensus}$ is the time necessary for the system to reach a conclusion that a node has failed and is an important performance metric for a gossip failure detection system. Failure detection services based on gossiping can also be flat or layered. Flat gossiping systems use only one group in their communication patterns. Layered gossiping systems use multiple groups, wherein the members of each group communicate mostly among themselves. Higher gossip layers handle communication within groups. Two-layer systems have nodes within each group in the lower layer (L1) take turns participating in the upper-layer gossip (L2). Gossiping performance in terms of consensus time is analyzed for a variety of gossiping communication patterns and for single-layered and multi-layered systems by Sistla, et al., in [9]. Layered gossip systems are demonstrated to be more scalable in terms of resource utilization by Sistla, et al., in [10].

Based on the findings of Sistla, et al., in [9] and [10], a two-layer gossip failure detection system was chosen and adopted to provide that part of the functionality of the CMS. That functionality was extended to also provide system load information at each managed node in the CMS as well as liveliness reports.

3. CMS description

The CMS is composed of four components. The normal operation of these parts is shown in Figure 1. The first component is the web-based client, from which user or administrative requests are issued. These queries can request status information for any subset of nodes within the managed system as well as the system loads on any subset of nodes. Furthermore, queries may also request that one or more nodes perform tasks.

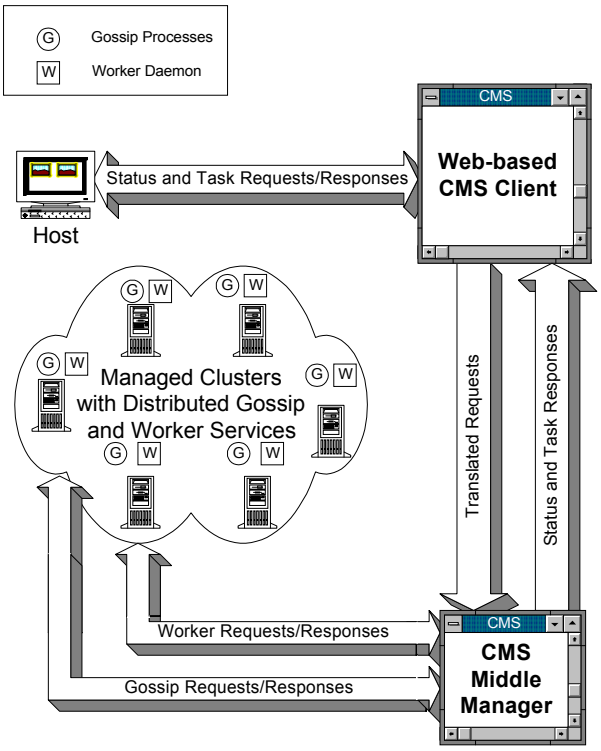

Figure 1. Operational hierarchy of the CMS

The second component of the CMS is the middle manager. The middle manager is called by the client program, and converts the requests of the users or administrators into the format understood by the lower-level cluster management tools being used. The middle manager can also be called directly by user-level applications to obtain needed information if desired rather than going through the client. This feature is particularly desirable for adaptive applications that condition the location where they execute on such important information as what nodes are operational and relatively lightly loaded. The ability to retrieve this information in this fashion has important implications for scheduling tools as well. Both centralized approaches to scheduling, such as IBM's *Loadleveler*, and application-level scheduling approaches can benefit from this information. For large clusters, the question is often not whether at least one node will be unavailable, or so heavily loaded as to effectively be so, but how many nodes are in such a state.

The third component of the CMS is the failure detection and consensus service based on gossiping. This service is implemented with multilevel random gossiping in two levels. The particular parameters for gossiping used by default by the CMS are T_{gossip} = 10 ms and $T_{cleanup}$ = 200 ms. These parameters result in a low consensus time in the detection of failed nodes. Increasing T_{gossip} and $T_{cleanup}$ would have the effect of both increasing the consensus time for failure detection and reducing the resource utilization of the CMS. Sistla, et al., discuss these tradeoffs in [9]. The failure detection service can be queried about the status of any node within the managed clusters directly if desired using UDP. Due to the distributed nature of gossiping, this query may be directed at any node within any cluster being managed.

The final type of component of the CMS is the worker daemon. A worker daemon runs on each node of the managed cluster, and serves to handle requests for the execution of tasks and to service queries about the system load on its node. The worker daemons were originally designed for job scheduling purposes and are described in more detail by Collins, et al. in [11]. Like the other components of the CMS, the worker daemons can be communicated with directly if desired using TCP. The implementation of the worker functionality in a single autonomous daemon per node insures scalability and the lack of a single point of failure for the services that it provides. In the future, provision to monitor the network resources being used at the node where the worker is operating is a planned addition to the CMS.

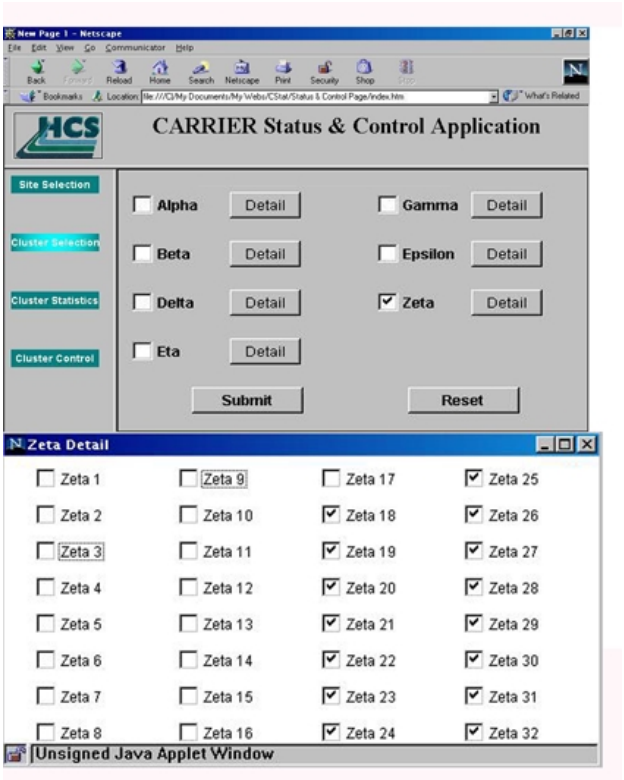

Figure 2. CMS screenshot of node selection

Figure 2 depicts a screenshot of the CMS client in operation. In Figure 2, the user has selected a subset of the Zeta cluster for later cluster management actions.

In Figure 3, the user has selected node liveliness as the data of interest. The CMS client has generated a report on the selected nodes.

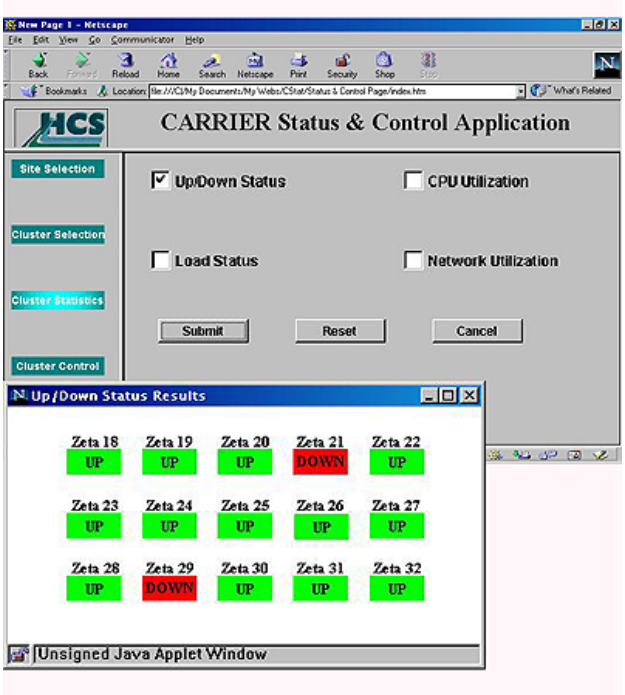

Figure 3. CMS screenshot for node liveliness report

4. Case study descriptions

A description of the two case studies is provided in this section. The experiments used to investigate each case study are also discussed.

4.1. Application-level overhead of the CMS

The first case study for the cluster management service was designed to determine the overhead of the service from the perspective of a representative set of applications. The applications chosen for this purpose were drawn from the Numerical Aerospace Simulation (NAS) benchmark suite. The first benchmark selected was the Class-A Conjugate Gradient (CG) benchmark. The CG benchmark solves an unstructured sparse linear system using the conjugate gradient method, and parallel execution is limited to numbers of processes that are integral powers of two. The second benchmark was the Class-W Simulated Computational Fluid Dynamics (SP) benchmark, which computes a finite difference solution of 3-D compressible Navier-Stokes equations. The SP benchmark is limited to numbers of processes that are perfect squares. The third benchmark is the Class-A Embarrassingly Parallel (EP) benchmark, which tabulates random numbers with a Gaussian distribution. Bailey, et al., describe all of the NAS benchmarks in more detail in [12]. These benchmarks, as will be seen in Section 5 to follow, span the range of levels of parallel utilization and scalability.

The machine set chosen for these experiments consisted of two clusters from our CARRIER system, each containing 32 dual-processor machines running Redhat Linux. The first cluster, Zeta, uses 733 MHz Pentium III processors each with a 256 KB L2 cache. Each machine in Zeta also has 256 MB of main memory. The second cluster, Delta, used 600 MHz Pentium III processors each with a 512 KB L2 cache. Each machine in Delta also has 256 MB of main memory. All machines in both clusters are connected using switched 100 Mb/s Fast Ethernet.

The first set of trials for each benchmark was conducted on nine nodes of Zeta with the entire system in a totally unloaded state. The sequential and parallel execution times for each of the three benchmarks is determined using the average run time of 30 trials in each case. The trials in this case study were conducted both in using one process per node (i.e., leaving the second processor on each machine idle), and using two processes per machine (i.e., using both processors of each dual-processor machine). Hereafter, the use of one process per machine will be referred to as the dual-CPU nodes, single process per node (DN-SPN) trials and the use of two processes per machine will be referred to as the dual-CPU nodes, two processes per node (DN-TPN) trials. The EP benchmark was evaluated for 1, 2, 4, 6, and 8 processes. The SP benchmark was evaluated for 1, 4, and 9 processes. The CG benchmark was evaluated for 1, 2, 4, and 8 processes. The particular machines used for each trial were held constant, as it was necessary to be able to detect small amounts of performance degradation due to the overhead of the CMS.

The second set of trials for the first case study was conducted after the cluster management service was activated. The cluster management service was started, managing the complete Zeta and Delta clusters. The particular parameters used for the gossip service were multilevel gossiping with 64 nodes divided into 8 groups, T_{gossip} set to 10 ms, and $T_{cleanup}$ set to 200 ms. Furthermore, the client for the CMS was set to generate a query once every 60 seconds. This query contained a request for the status information for all of the machines in both clusters as well as a request for system load information over the previous 60 seconds for each machine. Using *Ethereal,* a public domain packet-

capturing tool, it was determined that each node in the managed cluster used from 17-18 Kb/s of bandwidth to support the demands of the CMS. It was also determined that the bandwidth used was almost entirely (over 99.9%) due to the gossip-based failure detection and consensus service. After the CMS was started, the trials for each benchmark were conducted once more in precisely the same manner. Both the absolute levels of performance for each trial and the difference between the level of performance prior to the activation of the CMS and afterwards were determined.

4.2. Overhead effects of disabling the second CPU in SMP machines

The second case study was designed to test the hypothesis that the DN-SPN trials were benefiting from the fact that their unused processor was absorbing much of the overhead of all the auxiliary and support processes including the CMS. To test this hypothesis, each machine used in the execution trials (i.e. nine nodes of Zeta) had its second processor disabled. Once the second processor was disabled through the use of *lilo*, the Linux loader program, the single process per node portion of the trials from the previous case study was repeated. As the second processor had been disabled, the trials involving two processes per node were not revisited. The trials in this case study are hereafter referred to as uniprocessor nodes, single process per node (UN-SPN) trials. All other parameters for the experiments of this case study are the same as in the first case study. Once again, both the absolute levels of performance for each trial and the difference between the level of performance prior to the activation of the CMS and afterwards were determined.

5. Case study results

In this section, the results for each of the two case studies are presented. Each performance and overhead experiment described in Section 4 is analyzed and discussed.

5.1. Application-level overhead results for the CMS

The performance results prior to and after the activation of the cluster management service for the DN-SPN trials are shown in Figure 4. Examining Figure 4 yields a number of observations.

First, it is observed that the NAS EP benchmark shown in Figure 4(a) experiences excellent parallel utilization over the range of numbers of processes used in this study. Furthermore, the SP benchmark shown in Figure 4(b) achieves a level of parallel utilization somewhere between that of the EP and CG benchmarks. It is also obvious that the level of CMS overhead experienced in all the DN-SPN trials is very small. The performance differences will be amplified and examined in Figure 6 later in this section.

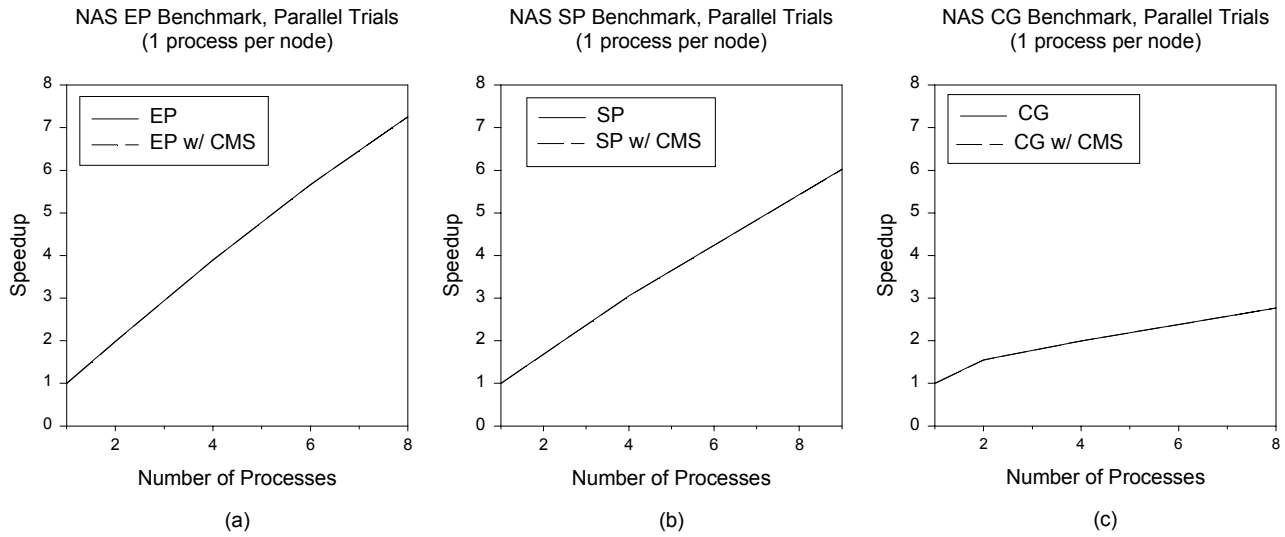

Figure 4. Performance impact of CMS overhead on NAS benchmarks (DN-SPN trials) for Class-A NAS EP (a), Class-W NAS SP (b), and Class-A NAS CG (c)

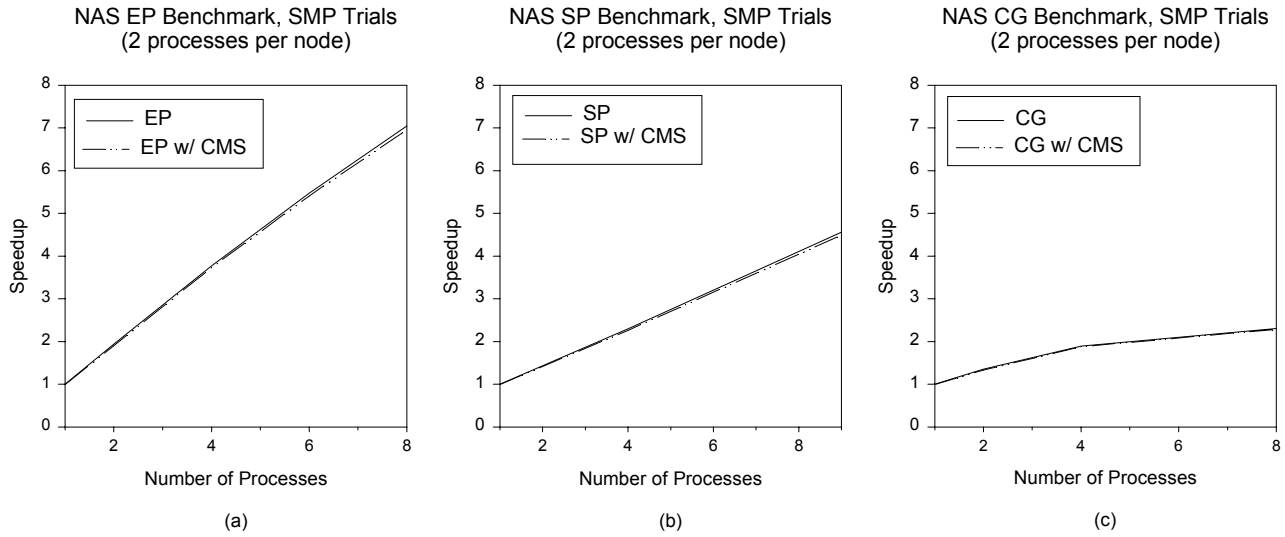

Figure 5. Performance impact of CMS overhead on NAS benchmarks (DN-TPN trials) for Class-A NAS EP (a), Class-W NAS SP (b), and Class-A NAS CG (c)

Figure 5 shows the performance results prior to and after the activation of the CMS for the DN-TPN trials. The speedup performance for the DN-TPN trials of Figures 5 are seen to be somewhat lower than their respective counterparts from the DN-SPN trials depicted in Figure 4. These particular results are not surprising. A dual-processor SMP often experiences slightly less speedup in parallel application performance than two otherwise identical single-processor machines. The difference comes about due to the increased contention for resources such as access to the memory and system buses present in a SMP configuration relative to a single-processor configuration. In addition, it is evident that the difference between performance prior to the activation of the manager and afterwards is greater for the DN-TPN trials. This difference is further investigated in Figure 6.

Figure 6 shows the trends of the CMS's overhead versus the number of processes being employed in the application. The overhead is quantified as the relative slowdown in execution time experienced after activation of the CMS versus the identical system prior to activation. In Figure 6, it is observed that the DN-SPN trials experience lower levels of effective overhead than do the DN-TPN trials. This observation is consistent with the smaller differences between the respective trendlines for the DN-SPN trials in Figure 4. The cause of this particular phenomenon is that the machines in the DN-SPN trials are largely able to hide the overhead of the CMS by making use of their unused second processor. The DN-TPN trials afford no such option, as both processors on each machine are in use, and thus experience a higher effective level of overhead. This overhead comes both in the form of contention for the CPUs, cache and memory contention, and in the necessity for occasional context switches. However, from the perspective of evaluating the CMS, the most important observation is that the CMS always causes less than a 2% degradation in application performance.

At this point it is appropriate to briefly discuss the confidence intervals of the results as these experiments have a stochastic character. The execution times are assumed to come from a normal distribution, and the standard error of the performance degradation taken to be equal to the sample standard deviation of that measurement divided by the square root of the sample size. Truncated normal distributions are often used to characterize execution times and, given the very small size of the sample standard deviation relative to the sample mean, neglecting that truncation is reasonable. Under these conditions, the standard error under normal conditions was generally found to be between 2.5% and 5% of the size of the performance degradation. Thus, there exists substantial statistical confidence that the results cited here represent a real effect and not simply a statistical accident.

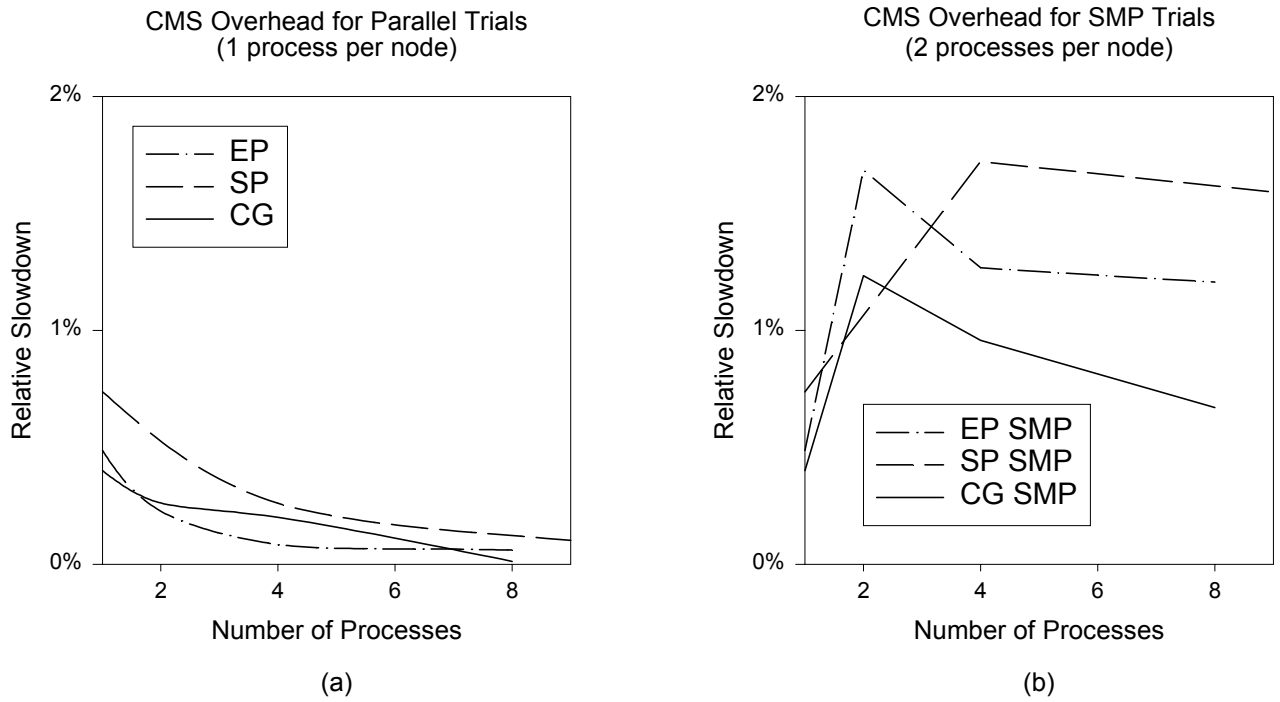

Figure 6. CMS overhead expressed as percent slowdown for DN-SPN trials (a) and DN-TPN trials (b)

In addition, it is observed that the application-level overhead generally decreases as the number of processes is increased for the application. This roll-off of the level of overhead is a very desirable feature, as it indicates that the system as a whole is highly scalable. The degree to which the overhead decreases with an increasing number of processes also appears to be inversely proportional to the parallel utilization of the application. Thus, the CG benchmark experiences the smallest levels of overhead as the number of processes increases and the EP and SP benchmarks experience the highest levels in Figure 6(b). This result is a consequence of the fact that the high levels of parallel utilization achieved by the EP and SP benchmarks require correspondingly high levels of CPU utilization. The CG benchmark, by contrast, requires a smaller level of CPU utilization for each process as the number of processes increase. It is thus able to overlap the service of the CMS and other auxiliary processes during the times when it is waiting on communications to a greater degree. The SP benchmark has parallel utilization levels and corresponding CPU utilization levels per process that are slightly less than the EP benchmark. The SP benchmark does have a considerably larger demand for memory than the EP benchmark. This larger memory footprint means that it experiences a larger slowdown due to CMS overhead as a result of contention for main memory and cache space. It should be noted that the SP benchmark only is valid for 1, 4, or 9 processes in this case study due to its requirement of a square number of processes. Thus, the intersection of the respective trendlines for the EP and SP benchmarks should be viewed as being an artifact of interpolation.

5.2. Overhead impact of disabling the second CPU in SMP machines

The performance results prior to and after the activation of the CMS are shown in Figure 7. All of the trials in this case study were conducted with the second processor disabled. An examination of Figure 7 shows that the difference between the levels of speedup performance due to the activation of the CMS is greater than in the previous case study. This effect is seen in the divergence between the trend lines for each benchmark and the corresponding trends for each benchmark with the manager in operation.

The average performance level for the UN-SPN trials of this case study prior to beginning operation of the CMS is slightly worse than the same DN-SPN trials in the previous case study. This effect is a direct consequence of the fact that the second processor of each node is not available to absorb the burden of much of the ordinary processes necessary to the normal operation of a UNIX machine. Such processes normally have low levels of resource utilization, but they do exert a small negative impact on application performance. For example, the

NAS EP benchmark running on a single processor experienced an average of a 0.41% slowdown when the second processor was disabled relative to its level of performance when the second processor was available to handle tasks not directly related to the application in question. While this effect is not large, it does indicate that case studies intending to compare parallel, single process per machine, vs. SMP, one process per CPU, performance should in some cases perform the parallel trials with the additional CPUs disabled. Otherwise, the comparison is likely to be slightly unfair to the SMP, as it would have to bear system overhead that the parallel trials would not. This adjustment may be highly important for experimental studies intended to precisely measure effects that are small in magnitude.

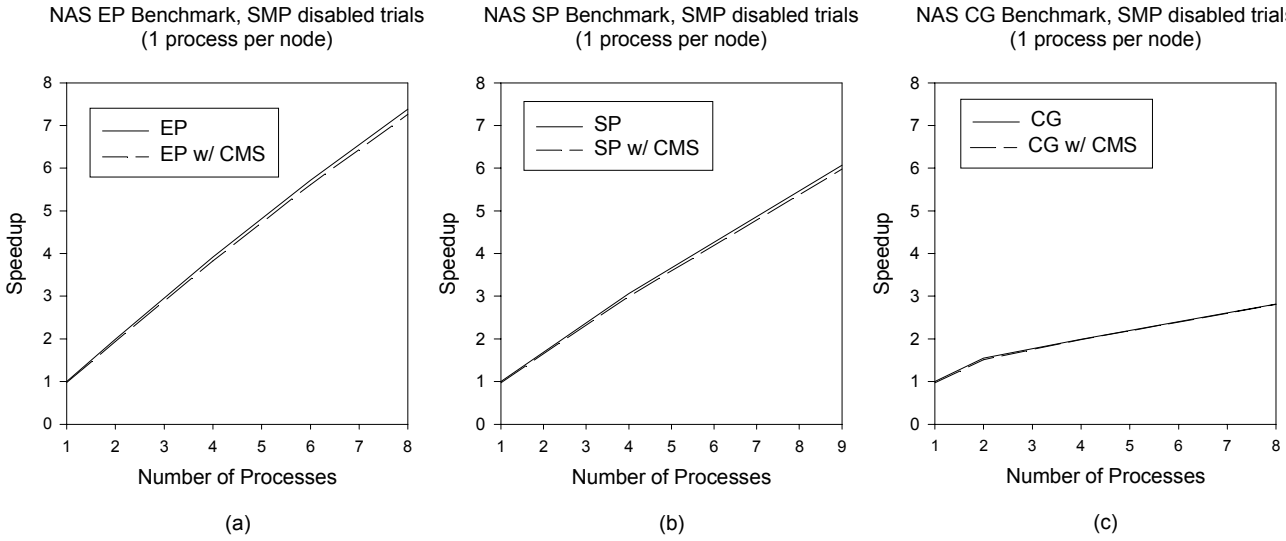

Figure 7. Performance impact of CMS overhead on NAS benchmarks (UN-SPN trials) for Class-A NAS EP (a), Class-W NAS SP (b), and Class-A NAS CG (c)

Figure 8 shows the relative overhead of the CMS as a function of the number of processes for each of the three benchmarks. As before, the overhead is quantified in terms of the percent slowdown of applications. An examination of Figure 8 reveals several insights. First, the trends of the overhead are much the same as in the previous case study. Overhead generally falls with an increasing number of participating processes. This effect is accounted for as before by the increased availability of free time on the CPU as the parallel utilization level decreases. The NAS CG benchmark's experienced level of overhead falls faster as the number of processes increases than either of the other two benchmarks. This observation is consistent with the previous case study. Another important observation here is that the absolute levels of overhead in this case study are higher than either the DN-SPN or the DN-TPN trials of the previous case study. The greater magnitude of application-level overhead in this case study is a result of the fact that the machines here have nowhere to divert the task of servicing the CMS or the auxiliary system processes. The DN-SPN trials of the previous case study had an entire free processor to use for this purpose, and accordingly were able to maintain the highest level of application performance. The DN-TPN trials were able to share the burden of the same overhead roughly evenly between their two processors. It is expected that SMPs with yet more processors (i.e., quad-processor SMPs of the same type) would experience even less effective overhead due to the ability to share it among their several processors.

An important observation from Figure 8 is that the overhead does not appear to significantly limit scalability, as it decreases with an increasing number of processes. In addition, the absolute level of application-level overhead is generally less than 3%. The measured level of overhead in this case study is consistent with the CPU utilization levels calculated by Sistla, et al., for failure detection systems with consensus using multilevel gossiping in [10]. This consistency is noteworthy as the CMS has also added the functionality of providing system load state information to the liveliness information distributed by the gossip service. Furthermore, the CMS has also incorporated the facility for the distribution of administrative directives.

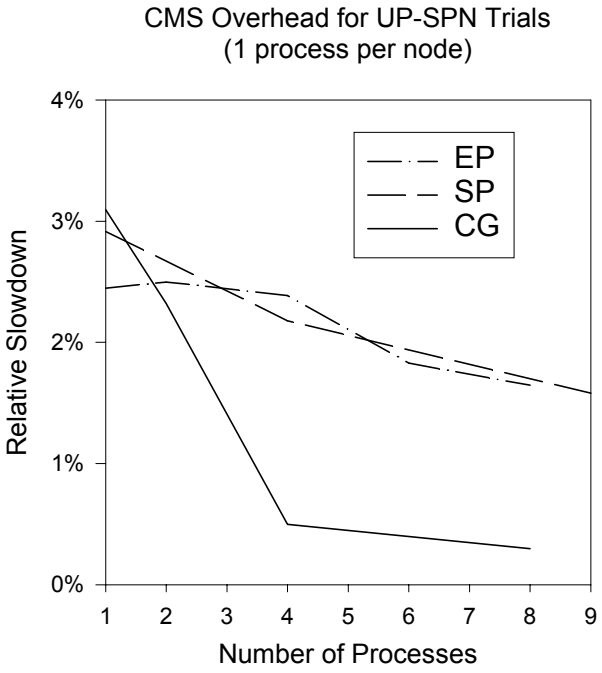

Figure 8. CMS overhead expressed as percent slowdown for UP-SPN trials

6. Conclusions

In this paper, an extension of a gossip service for fault detection is developed to provide system load state information for the managed nodes as well as node liveliness. In addition, the capability to issue administrative directives in a centralized fashion is also implemented. The fusion of these services and the client and application interfaces forms the basic CMS, which will be extended through the addition of more services (e.g., network resource monitoring) in the future. Furthermore, the scalability and overhead imposed by the CMS based on gossiping was determined and evaluated.

This contribution shows the feasibility of managing large networks using a gossip-based approach while surrendering very little parallel and sequential application performance. CMS overhead was in all cases not more than about 3% from the perspective of application slowdown. This modest level of overhead can be further reduced by increasing T_{gossip} or polling the system less frequently if a lower level of management performance can be tolerated. The downward trend in application-level overhead for the CMS also indicates that it is unlikely to significantly interfere with parallel jobs on clusters using large numbers of processors. Furthermore, this research develops a framework for the integration of further services for cluster management, such as the sampling of network loads. These cluster management capabilities lay a portion of the groundwork for the more intelligent and efficient scheduling of heterogeneous cluster resources.

A comparison of the DN-SPN approach and the DN-TPN approaches over the set of three NAS benchmarks showed that the use of the DN-TPN approach was more affected by the overhead of the cluster management service. A further evaluation of the overhead of the CMS was performed by repeating the one process per machine experiments with the second processors disabled. These experiments show that the overhead is most significant when there are no available additional resources with which to share the overhead or onto which to shift it.

It is clear that part of the advantage of a dual-processor SMP over an otherwise identical uniprocessor machine is the ability to execute applications relatively unhindered by the sort of auxiliary support processes common in UNIX environments. Only when the additional processors are disabled is the comparison truly fair to the SMP.

There are several possible directions for future research. One direction is the scalability analysis of the overhead imposed by the cluster management of larger networks. Another possible direction is the integration of further services, such as uniform clock synchronization, into the CMS using the same general approach. Yet another possible area for future research is the application of the services implemented by the CMS to the intelligent adaptive scheduling of applications. The CMS could be used to refine the prediction of the expected execution time of applications when the scheduling algorithms are designed to make use of such input.

7. Acknowledgements

The support provided by the NSA is acknowledged and appreciated, as are equipment grants from Nortel Networks and Intel that made this work possible.

8. References

1. B. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Reading, MA: Addison-Wesley, 1989.
2. G. Pfister, *In Search of Clusters*, *Second Edition* Upper Saddle River, NJ: Prentice Hall, 1998.
3. U. Black, *Network Management Standards, Second Edition*, New York: McGraw-Hill, 1995.
4. T. Aamodt, "Design and Implementation Issues for an SCI cluster configuration system," *Proc. of SCI Europe-1998*, Bordeaux, France, September 28-30 1998.
5. R. Buyya, "PARMON: A Portable and Scalable Monitoring System for Clusters," *Software—*

Practice and Experience, Vol 30 n 7, Jun. 2000, pp. 723-739.

6. R. Van Renesse, R. Minsky, and M. Heyden, "A Gossip-style Failure Detection Service," *Proceedings of IFP International Conference on Distributed Systems Platforms and Open Distributed Processing Middleware 1998*, Lake District, England, Sep 15-18, 1998.
7. M. Burns, A. George, and B. Wallace, "Simulative Performance Analysis of Gossip Failure Detection for Scalable Distributed Systems," *Cluster Computing*, Vol. 2, No. 3, 1999, pp. 207-217.
8. S. Ranganathan, A. George, R. Todd, and M. Chidester, "Gossip-style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters," *Cluster Computing*, accepted and in press.
9. K. Sistla, A. George, R. Todd, and R. Tilak, "Performance Analysis of Flat and Layered Gossip Services for Failure Detection and Consensus in Scalable Heterogeneous Clusters," *Proc. of IEEE Heterogeneous Computing Workshop (HCW) at the Intnl. Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, CA, April 23-27, 2001.
10. K. Sistla, "Experimental Analysis of Flat and Layered Gossip Services for Failure Detection and Consensus," MSECE Thesis, Dept. of Electrical and Computer Engineering, University of Florida, Gainesville, FL, February 2001.
11. D. Collins and A. George, "Simulation of Parallel and Sequential Job Scheduling in Heterogeneous Clusters using Software-in-the-Loop," submitted to *Simulation*, February 2001.
12. D. Bailey, J. Barton, et al., "The NAS Parallel Benchmarks," Technical Report RNR-91-02, Moffet Field, CA: NAS Systems Division, NASA AMES Research Center, 1991.