Performance Monitoring on an HPVM Cluster

Geetanjali Sampemane
geta@csag.ucsd.edu

Scott Pakin
pakin@cs.uiuc.edu

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W Springfield Ave, Urbana, IL 61801

Andrew A. Chien
achien@cs.ucsd.edu
Department of Computer Science and Engineering
University of California at San Diego
9500 Gilman Drive, #0114, La Jolla, CA 92101

Abstract

Clusters of workstations are becoming popular platforms for parallel computing, but performance on these systems is more complex and harder to predict than on traditional parallel machines. Hence, performance monitoring and analysis is important for understanding application behavior and improving performance. We present a performance monitor for HPVM, a high-performance cluster running Windows NT. The novel features of our monitor are: an integrated approach to performance information, a (software) global clock to correlate performance information across cluster nodes and leverage of Windows NT performance monitoring facilities. We discuss the design issues for this tool, and present results of using this tool to analyze the performance of a cluster application.

Keywords: cluster performance monitoring

1 Introduction

Clusters of workstations or PCs are now an attractive alternative to massively parallel processors (MPPs) for parallel computing. They have good price/performance characteristics, and recent developments in high-speed interconnects help these clusters get aggregate performance comparable to supercomputers at a fraction of the price.

However, there are problems with treating clusters as traditional parallel machines. As systems get increasingly complex, the factors that can affect application performance also increase—OS task scheduling, network behavior and software messaging layers can each influence application performance dramatically. Since we are interested in parallel-processing clusters where the main goal is high performance, performance monitoring becomes an essential tool for obtaining good application performance. A systematic and integrated approach to performance monitoring is essential. This is the problem we address.

We have designed and implemented a performance monitor for an HPVM (High-Performance Virtual Machine) cluster consisting of 64 dual-processor Pentium nodes running Windows NT and HPVM 1.9 software, connected with high-speed interconnects. HPVM supports standard messaging interfaces such as MPI implemented over Illinois Fast Messages (FM) [1], a low-latency, high-bandwidth messaging layer.

The main goal for the performance monitor was to assist performance tuning of applications on the cluster. A secondary goal was to evaluate the performance of the various messaging layers and provide information that could help tune the design.

The performance monitor usually runs in online mode and allows the set of monitored parameters to be dynamically configured. Software time synchronization provides a global

timebase to correlate information from the different cluster nodes. The performance monitor also makes use of the support for performance monitoring provided by Windows NT.

The unique features of our performance monitor are the ability to monitor cluster applications online, a software global clock to correlate performance information across the cluster and good integration with the Windows NT performance monitoring facilities. The ability to correlate system performance information with information from the HPVM system allows us to quickly detect sources of performance problems which are not obvious from looking at them separately.

The remainder of this paper is organized as follows: Section 2 provides background on the issues involved in performance monitoring and the HPVM cluster. Section 3 describes the design and implementation of our performance monitor. Section 4 presents the results of using the performance monitor. We then discuss the results in the context of related work and finally conclude.

2 Background

2.1 Performance Monitoring

System performance can be monitored at a variety of levels. Different types of information are available at different levels, and there is often a trade-off between the type and amount of information obtained and the overhead imposed.

Performance monitoring in hardware often imposes the lowest overhead, but information available at this level is often too low-level and hard to relate to application behavior. Hardware modifications on each node for performance monitoring may also be undesirable while using commodity components.

Even in software, different levels provide different types of information. Aggregate information about system resource utilization may be available at lower levels, while monitoring at the application level may provide information that is easier to relate back to the programming model for analysis.

A decentralized approach is essential for scalability as clusters grow in size. A unified view is important, so the challenge is to collect and synchronize information from the various nodes to present an overall picture.

2.2 HPVM

The work described in this paper was done in the context of the HPVM [2] project. The goal of the HPVM project is to create a software infrastructure to enable high-performance parallel computing on PC clusters, leveraging commodity hardware and software.

The HPVM cluster at UCSD consists of 64 Pentium III nodes connected by Myrinet, Giganet and Fast Ethernet. The cluster nodes run the Windows NT operating system, and HPVM 1.9 software. Illinois Fast Messages [1] provides a high-bandwidth low-latency user-level messaging layer that can use Myrinet, Giganet or shared memory (for intra-node communication), and HPVM supports standard supercomputing APIs such as MPI [3], which makes it easy for users to port their parallel applications to run on an HPVM cluster. Monitoring application performance is even more important in the case of such "legacy software", to evaluate whether the design decisions made for different architectures and software environment are still appropriate.

2.3 Windows NT

Windows NT provides a standard performance API that enables applications and other system components to provide performance information about themselves. NT also makes available a variety of "system" information, such as processor utilization and paging behavior. All such performance information can be queried using a standard interface, both locally and from other machines on the network.

All performance information is maintained in the form of "performance objects" that contain a set of "counters" that are updated by the application or OS to contain the performance information. NT also provides two interfaces to query this performance information—the

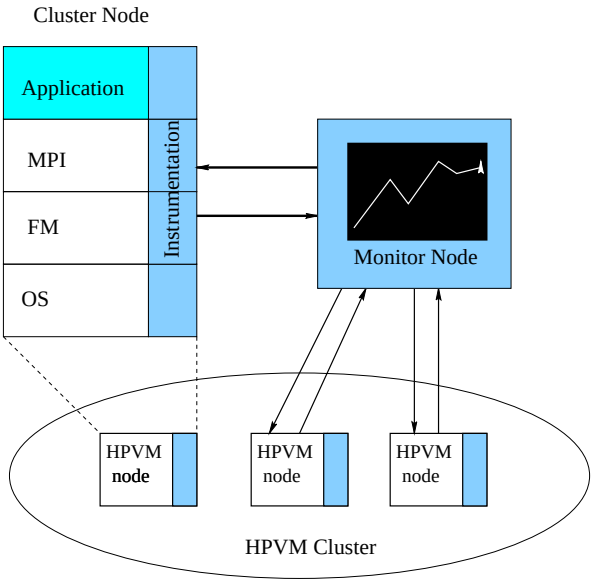


Figure 1: Architecture of HPVM Performance Monitor

Registry interface and the Performance Data Helper (PDH) interface—that can be used to develop custom tools for collecting performance information. The standard NT Performance Monitor utility (`perfmon`) can collect all this information and either display it graphically or log it to a file.

3 Design

In this section, we describe the interesting aspects of the design, implementation and user interface of the HPVM performance monitor.

3.1 Design Objectives

The main design objective for the HPVM performance monitor was to provide a uniform interface for performance information from all parts of the system that affect performance. Good scalability and low overhead were obviously important aspects to consider.

To achieve these objectives, the system was designed as shown in Figure 1. Each node is instrumented to capture a variety of performance information. This information is maintained locally in the form of NT performance objects. A central "monitor" node queries cluster nodes periodically to collect and process this information. Per-node overhead is thus limited to instrumentation, and to sending performance data when requested. The Fast Ethernet network is used to transfer this performance information, to minimize impact on the application, which uses the faster Myrinet or Giganet networks.

Another decision was to make the monitoring process as transparent to the user as possible—i.e. no modification of the application is required. Data is collected from fixed instrumentation points in the operating system and the HPVM system. The user need only link his application with the instrumented version of the FM and MPI libraries.

3.2 Integration with Windows NT

We decided to use the Windows NT performance interface, because this easily gives us access to system performance information and because it allows us to leverage the available NT tools for viewing performance information. Designing the instrumentation therefore reduces to deciding upon a set of "performance objects" and creating a set of counters for each object that convey all interesting information in a meaningful way. Thus we have uniform access to performance information about the base operating system alongside additional information about HPVM software, that can all be displayed with standard NT performance monitoring tools. This enables easier correlation of operating system behavior with that of the HPVM system, which is often critical for accurately diagnosing problems.

Performance objects are updated during the application execution, so information is available in real-time. Being able to adaptively modify the set of information being collected based on current behavior is also useful, since it allows the user to "zoom in" on potential sources of problems when they appear.

3.3 HPVM Performance Objects

Figure 1 shows the software layers in an HPVM system. Since NT already collects information about the OS, we instrument the Fast Messages and the MPI layers. FM provides a high-performance data delivery service to

higher-level messaging layers like MPI, so the FM performance object provides information mainly about data transfer. Apart from aggregate amounts of data sent and received by each node, we also keep track of message size distributions. Information at this level is useful for detecting problems with the network, or things like network contention. At the MPI layer, we collect two kinds of information. We use the MPI Profiling Interface to collect information about the MPI library in an MPI performance object. Another MPID performance object maintains lower-level information about the MPI implementation, such as message size distributions and current lengths of various internal message queues, that can indicate problems with buffer management or application communication patterns.

Each of these performance objects—FM, MPI and MPID—can be monitored independently of the other. One counter, the DebugCounter, is visible to the application programmer. The value of this counter has no specific meaning, and can be used by the application programmer to indicate occurrence of certain events.

3.4 Data collection and reduction

Performance information on each node is stored as a shared memory object, and can be accessed remotely using the NT performance API. The monitoring frequency is decided by the monitor node. The volume of performance data that needs to be transferred depends on the number of counters and is a few hundred bytes per counter. We use the Fast Ethernet network, not normally used by parallel applications on HPVM, to collect performance information from cluster nodes onto a monitor node.

3.5 Time Synchronization

A global timebase is essential to correlate information obtained from different nodes. However, clusters rarely have a globally synchronized clock. Each node has a local clock, and all performance information is accompanied by a local timestamp. Since the local clocks are not coordinated in any way, we use a variant of Cristian's algorithm [4] to synthesize a global clock by calculating offsets of different node clocks from the monitor clock.

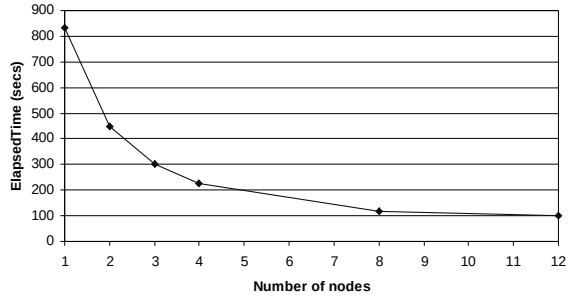
3.6 Usage

The application has to be linked with instrumented versions of the FM and MPI libraries to enable monitoring. The user runs our `timestamps` program over all nodes before and after the application to collect timestamps from all machines and estimate local clock offsets from the global (monitor) clock, and specifies the list of counters and optionally a list of hosts and/or processes to monitor, and an interval at which to collect information. While the application is running, the monitor collects the values of these counters from each of the nodes. Counters can be selected individually, or grouped into sets that provide information about higher-level categories, such as "FM data transfer" or "MPI collective communication".

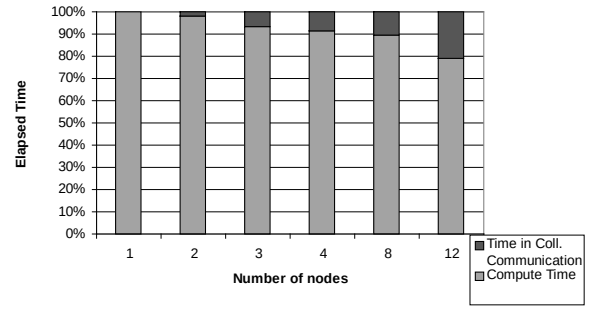
4 Application Example

We have used the performance monitor to observe various applications running on our cluster. We present the results of using the performance monitor on `cholesky`, an MPI program to perform Cholesky factorization. The program heavily uses collective communication (broadcasts and reductions), which is interesting for performance analysis, because every operation is limited by the slowest node. We illustrate how the performance monitor can be used to diagnose performance problems in the network layer as well as those caused by problems in the host operating system. The ability to view all system information together is essential to diagnose all these.

The monitor itself could theoretically affect the application performance in two ways: contending with the application for CPU and for the network. To measure monitoring overhead, we ran the `cholesky` program with and without instrumentation, and measured the total



(a) Run-time speedup



(b) Communication time as a fraction of normalized runtime

Figure 2: Cholesky Speedup

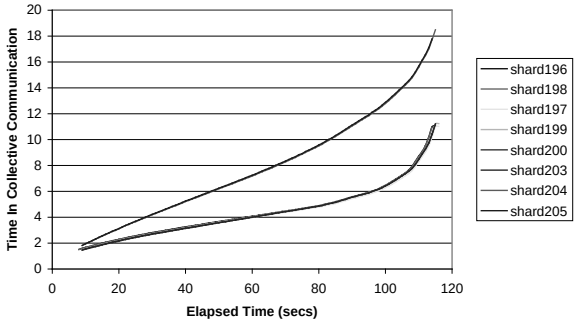


Figure 3: Time in collective communication

runtime. The overhead was negligible (between 0 and 5 seconds on runs of up to 800 seconds)—less than the measurement error.

Figure 2 shows the speedup of the `cholesky` program as the number of nodes increases from 1 to 12. While the total execution time decreases from 831 seconds to 102 seconds (Figure 2a), the fraction of time spent in collective communication (Figure 2b) increases from near–zero for the single–node case to around 20% of the runtime for the 12 node case. This indicates that the collective communication performance becomes more important as the application scales up.

We now show an example illustrating usage of the performance monitor. When a user notices a communication-intensive application running poorly, he could start by looking at the network performance. If that seems normal, then it is time to look at the "host" information. One "slow" node could suggest problems like stray competing processes, which might need to be killed or moved. The ability to look at the entire system and correlate the bad performance with the causes is valuable.

Figure 3 shows an 8-node `cholesky` process on the cluster, and the amount of time spent in the collective communication phase by each node. The NT "Process Elapsed Time" counter measures how long the application has been running, while the MPI "TimeInColl" counter measures the time spent in the collective communication routines. The time spent in collective communication varies from around 10% of the total runtime for some nodes (≈ 11 seconds out of a total runtime of 109 seconds) to a high of close to 20% of the total runtime (≈ 19 seconds out of 109). Thus some nodes spend a significantly larger fraction of the time in the collective communication routines. Typically the node that spends the *least* amount of time in the collective communication routines is the slowest—but all nodes have to wait until it is done.

We also monitored a variety of system counters to monitor what system resources the application used. The Process object allowed us to observe that `cholesky` had nearly exclusive use of the CPU throughout its run, on all the nodes it was running on. The memory behavior was very uniform—the working set of the program fit in memory and stayed there, as shown by a low system page fault rate. This information is not shown here because the values largely remain constant during the application run, but it was useful as a baseline to compare against when diagnosing bad performance.

Our first test was to see if we could identify network bottlenecks. Since our cluster is multi-user, we often have different applications running on different subsets of it. Since they sometimes share network switches, we decided to see

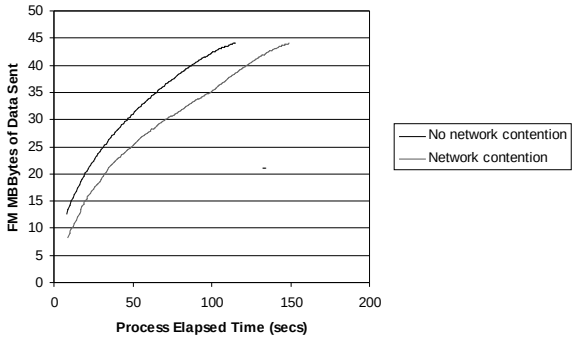


Figure 4: FM data transfer with and without network contention

how two such applications would affect each other. So during the run of `cholesky`, we had a different and very network-intensive application running on a part of the cluster. (This application was running on separate hosts from the ones on which we ran `cholesky`, but shared switches with some of them). The added network contention significantly increased the run-time of `cholesky`, from 109 seconds to 150 seconds. Looking at the system counters for processor utilization and memory behavior shows that there is no contention on the host—no increase in the number of context switches or page faults, and no decrease in the application's share of CPU—so we conclude that the problem is at the network. This is confirmed by looking at the data transferred at the FM layer. Figure 4 shows the time history of FM data transfer in each of the two runs—data transfer in the second run (with network contention) is much slower. Without having access to host information, it would be harder to eliminate operating system problems (like scheduling or paging problems) as a source of slowdown.

Our next example shows a case where the problem was indeed at the host level, and unrelated to HPVM. We started a competing process on one of the nodes during the run of `cholesky`. The `cholesky` program was slowed down and took around 250 seconds to complete (as compared to the 109 seconds without any host or network contention). Looking at processor utilization showed that one node received only around 60% of the CPU, as opposed to nearly 100% that the other nodes get, thus pointing to CPU contention as the problem.

This example shows how the ability to correlate OS and messaging layer is useful in pinpointing the source of performance problems.

5 Related Work

Performance monitoring is always an important tool for obtaining maximum performance; various approaches have been taken to achieve this.

The SHRIMP Performance Monitor [5] is closest to our work; it also aims to monitor performance of applications on a cluster. However, it approaches the problem at a lower level, with two solutions–a hardware monitor, and instrumentation of the Myrinet firmware to collect information about communication latencies. The SHRIMP project has also implemented an adaptive clock synchronization algorithm [5]. We did consider instrumenting the Myrinet interface, but concluded that it would provide no further information unless we had even tighter clock synchronization than we currently do.

The Pablo [6] toolkit focuses on portable performance data analysis and source code instrumentation and data capture. Our performance monitor trades off portability for the ability to obtain lower-level information about our specific platform.

Paradyn [7] is a tool designed for long-running parallel and distributed programs and large systems. It works by dynamically instrumentating running parallel programs, and automates the search for performance bottlenecks with a model of performance bottlenecks and their probable causes. The dynamic instrumentation allows it to control instrumentation overhead by stopping when the overhead reaches a threshold. This would be an interesting feature to add to our monitor.

The Digital Continuous Profiling Infrastructure [8] is a sampling-based profiling system that works on unmodified executables and has low overhead. It was designed to run continuously on production systems. Similarly, one of the goals for our performance monitor has

been to have the overhead low enough to run it all the time.

6 Summary

This paper presented a low-overhead performance monitor for an HPVM cluster. Our focus was a monitor that could scale to large clusters and that can integrate performance information from all parts of the system. A software time synchronization scheme provides a global timebase to correlate events on the different cluster nodes. We presented the results of using the performance monitor to analyze the `cholesky` application and demonstrated how the tool can be used to diagnose problems at different levels of the system.

Our performance monitor takes advantage of the Windows NT support for performance monitoring. This allows us to access all system performance information in a uniform manner and present it in a unified format. Another advantage of using the Windows NT performance data format is the ability to leverage available tools for displaying this information. Performance information about a cluster application can either be viewed in real-time, as the application executes, or logged for off-line analysis.

Our decision to use the NT Performance API proved extremely useful; many of the performance problems we found could be diagnosed from the "system" information, and some of them would have been invisible from just looking at the application and network. Having uniform access to operating system and application information allows us to clearly see the interactions across the system levels.

7 Acknowledgements

The research described is supported in part by DARPA orders #E313 and #E524 through the US Air Force Rome Laboratory Contracts F30602-99-1-0534, F30602-97-2-0121, and F30602-96-1-0286. It is also supported by NSF Young Investigator award CCR-94-57809 and NSF EIA-99-75020. It is also supported in p art by funds from the NSF Partnerships for Advanced Computational Infrastructure—the Alliance (NCSA) and NPACI. Support from Microsoft, Hewlett-Packard, Myricom Corporation, Intel Corporation, Packet Engines, Tandem Computers, and Platform Computing is also gratefully acknowledged.

We are grateful to Mike Heath at the Center for Simulation of Advanced Rockets (CSAR) for providing the Cholesky code.

References

[1] S. Pakin, V. Karamcheti, and A. A. Chien. Fast Messages: Efficient, portable communication for workstation clusters and MPPs. *IEEE Concurrency*, 5(2):60–73, April-June 1997.

[2] A. A. Chien, M. Lauria, R. Pennington, M. Showerman, G. Iannello, M. Buchanan, K. Connelly, L. Giannini, G. Koenig, S. Krishnamurthy, Q. Liu, S. Pakin, and G. Sampemane. Design and evaluation of an HPVM-based Windows NT supercomputer. *International Journal of High Performance Computing Applications*, 13(3):201–219, Fall 1999.

[3] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, June 1995. Version 1.1. Available from `ftp://ftp.mcs.anl.gov/pub/mpi/mpi-1.jun95/mpi-report.ps.Z`.

[4] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158, 1989.

[5] C. Liao, M. Martonosi, and D. W. Clark. Performance monitoring in a Myrinet-connected Shrimp cluster. In *Proceedings of the SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT)*, August 1998.

[6] 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 A. Skjellum, editor, *Proceedings of the Scalable Parallel Libraries Conference*. IEEE Computer Society, 1993.

[7] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irwin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tools. In *IEEE Computer*, pages 37–46, Nov 1995.

[8] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. A. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems*, pages 357–390, Nov 1997.