Technical Report
NWU-CS-01-6
July 17, 2001

Windows performance modeling and data reduction using WatchTower and Argus

Michael W. Knop **Praveen K. Paritosh** **Peter A. Dinda** **Jennifer M. Schopf**

Abstract

We describe and evaluate WatchTower, a system that simplifies the collection of Windows performance counter data, and Argus, a statistical methodology for evaluating this data and reducing the sheer volume of it. This is especially important when monitoring the performance of clusters for high performance computing. WatchTower's overheads are comparable to those of Microsoft's perfmon tool, while, unlike perfmon, it supports higher sample rates and is easily embedded into other software. Argus can reduce the behavior of a large number of raw performance counters into a few composite counters, or it can select a subset of the original counters that are statistically interesting.

We are currently integrating WatchTower and Argus into a single system which we believe will be useful in a number of scenarios.

Windows Performance Monitoring and Data Reduction using *WatchTower* and *Argus* *Extended Abstract*

Michael W. Knop   Praveen K. Paritosh   Peter A. Dinda   Jennifer M. Schopf
{knop, paritosh, pdinda, jms}@cs.northwestern.edu
Department of Computer Science
Northwestern University
1890 Maple Avenue
Evanston, IL 60201

July 17, 2001

Abstract

We describe and evaluate WatchTower, a system that simplifies the collection of Windows performance counter data, and Argus, a statistical methodology for evaluating this data and reducing the sheer volume of it. This is especially important when monitoring the performance of clusters for high performance computing.

WatchTower's overheads are comparable to those of Microsoft's *perfmon* tool, while, unlike *perfmon*, it supports higher sample rates and is easily embedded into other software. Argus can reduce the behavior of a large number of raw performance counters into a few composite counters, or it can select a subset of the original counters that are statistically interesting. We are currently integrating WatchTower and Argus into a single system which we believe will be useful in a number of scenarios.

1 Introduction

Clusters are becoming the standard platform for high performance computing. While most clusters run UNIX variants, PCs running variants of Microsoft Windows are becoming increasingly common. For example, Woodward's group at Minnesota uses Windows clusters for its PowerWall visualization work because of the lack of Linux drivers for the wide range of graphics cards needed [44].

Windows clusters have made several advances in the last few years. Very large Windows clusters have been built [2], and basic tools for message passing [36, 12], queuing [26] and resource management [18, 3] have been ported. However, performance monitoring, a basic service needed in high performance computing, is still in its infancy compared to UNIX.

We describe and evaluate *WatchTower*, a system that simplifies the collection of Windows performance counter data, and *Argus*, a statistical methodology for evaluating this data and reducing the sheer volume of it. We are currently combining these two tools, and this paper supports the feasibility of the approach.

WatchTower provides easy access to raw performance counters with comparable overhead to Microsoft's *perfmon* running in non-GUI mode. Furthermore, WatchTower is designed to be easy to embed into other monitoring systems.

Argus reduces the behavior of a large number (currently about 250, but potentially over a thousand) of raw performance counters to a few composite counters which nonetheless still capture the overall activity of the machine and its interactive user. It also selects a subset of the original counters that contains only the most significant uncorrelated counters.

These tools allow us to compare and study user activity across machines and over time. We envision a variety of usage scenarios such as: *platform and application signatures* [31, 40, 34], *scheduling* [33], *adaptation* [5] *resource management* [27], *user profiling*, *intrusion detection* [13], and *troubleshooting* [11].

These scenarios motivate the combined WatchTower/Argus system. We argue that the combination will be feasible in terms of its overhead. The system will give users and other tools easy online access to both raw and data-reduced Windows performance information. The results of our evaluation of WatchTower and Argus are very promising.

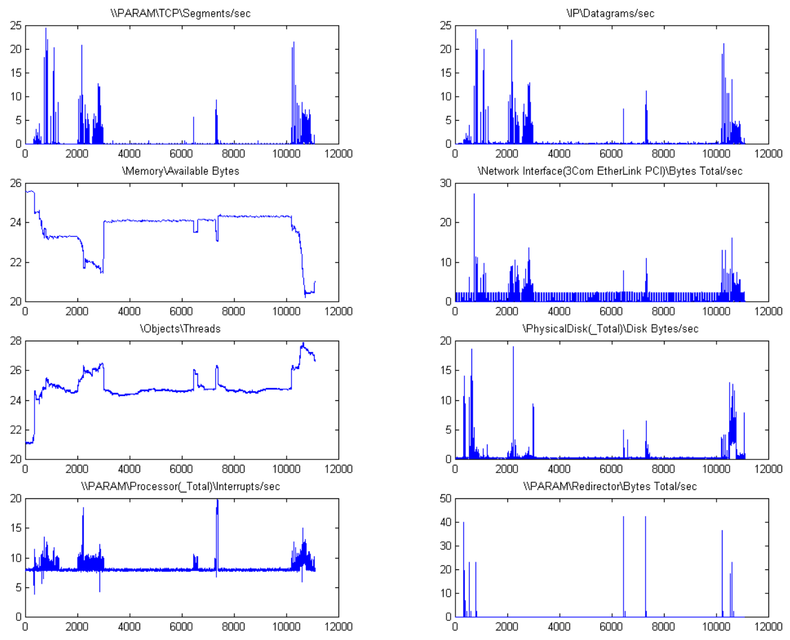

Figure 1: Representative logs of interesting counters during web browsing

2 WatchTower

WatchTower is an unobtrusive performance monitoring service for the Windows 2000 and NT operating systems. By "unobtrusive" we mean that the user remains unaware that the computer they are using is being monitored. WatchTower uses few resources and it can be configured to start and stop automatically, requiring no user interaction and displaying no windows once installed. WatchTower can also be used as library.

2.1 Measuring Performance in a Windows Environment

Windows contains a measurement infrastructure that is made visible to applications by means of roughly one thousand performance counters that together reflect the current state of the system. The exact number of counters varies depending on the system's configuration. Counters are arranged in a simple two-level hierarchy. The first level of the hierarchy is the physical or system *performance object* being monitored while the second is the *performance counter*, which is an attribute of the object. Figure 1 plots some of the more interesting counters while web browsing as an example.

Counters are stored and updated in the Windows registry, and can be accessed by a registry API. However, working directly with the registry is complex, and thus Microsoft also provides a more abstract API called the Performance Data Helper (PDH). This API handles the accessing of the counters in the registry and converting their raw values into numbers with appropriate units. PDH is the basis for both *perfmon* and WatchTower. Figure 2 shows an overview of how these parts interact.

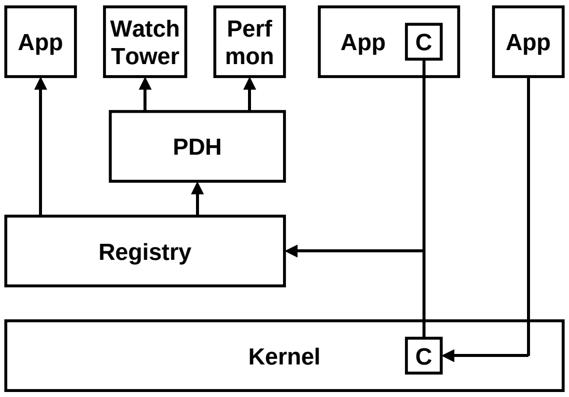

Figure 2: Windows Performance Monitoring

2.2 Perfmon

One application that uses PDH is Microsoft's Performance Monitor, also referred to as *perfmon* [21, 22]. *Perfmon* can operate entirely in the background, hidden from the interactive user. It is configured through an HTML-like script. However, *perfmon* has several deficiencies that limit its long-term logging capability and usefulness, namely granularity, over-writing files, and adaptability. This significantly affects its ability to provide adequate logging for high performance cluster computing in this environment.

The finest measurement granularity *perfmon* supports is one second, which is inadequate for many uses of windows logging data in a high performance computing system. In contrast, WatchTower supports a granularity of 10 ms (limited by the Windows timer granularity) and a corresponding peak rate of 100 Hz. *Perfmon* also overwrites the last log file after even graceful reboots. This drawback makes *perfmon* undesirable for collecting long term traces of machine behavior. WatchTower avoids this problem by starting a new log file every hour and after system start up. Finally, *perfmon* is difficult to incorporate into other systems or extend with new functionality (for example, data reduction using Argus). WatchTower can be used as simple C++ library and thus can be embedded into other programs trivially.

2.3 WatchTower Architecture

WatchTower is comprised of two main code components: service code and the WatchTower library, which uses PDH. WatchTower is based on the standard Windows infrastructure (services, the registry, error handling, etc.) and supplies programmatic access to these tools that non-Windows developers will find familiar to UNIX systems.

WatchTower is a service, similar to a UNIX daemon, that is run by the kernel without user interaction. It is set to start automatically upon system start up. While the code for running as a service is not complex, understanding the timing issues involved is. Certain sections of the service code must start within strict time limits, otherwise the service manager of the kernel will drop the process. The code we used is based on two sources [39, 38], in addition to the examples in the Platform SDK [20].

The most interesting part of WatchTower is the library that interacts with PDH. It is concerned with opening queries and logs, adding counters to the list to be sampled, and retrieving those counters' values at a periodic rate. As a library, this portion of the code can be easily included into other projects. We based the WatchTower library on code and application examples in the Platform SDK [20].

WatchTower can be easily embedded into any number of systems or applications that need a performance monitor. For example, it can be used in stand-alone command-line application to be run as needed, as part of a service (as it is now) to insure uninterruptible performance monitoring, to feed custom GUIs that present the data in new ways, to drive a closely coupled analysis tool such as Argus, or to provide sensors for systems such as Remos [19], NWS [43], and RPS [6, 7].

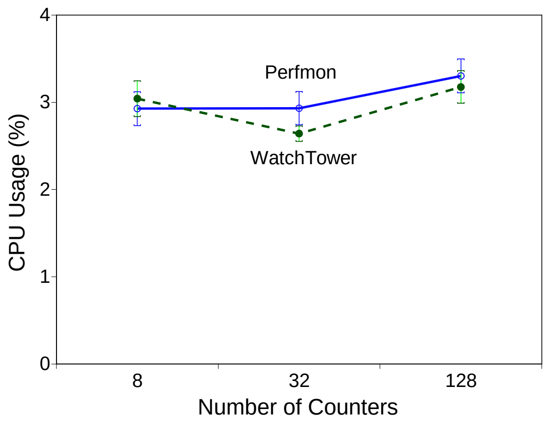

Figure 3: Overhead vs. counters at 1 Hz for Perfmon and WatchTower

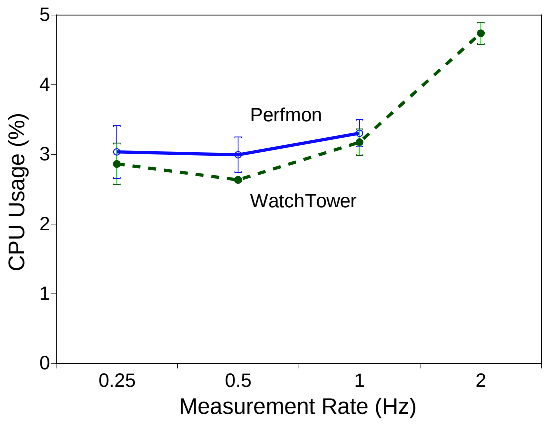

Figure 4: Overhead vs. rate with 128 counters for Perfmon and WatchTower

2.4 Overhead

WatchTower's overhead is similar to that of *perfmon*. Figure 3 shows the overhead (as % of CPU time used) of *perfmon* and WatchTower as a function of the number of counters and Figure 4 as a function of the measurement rate. With 128 counters, WatchTower can monitor at a peak rate of 17 Hz on a 650 MHz Pentium III machine.

Currently, both tools use the disk in exactly the same way (a function of PDH). However, the memory footprint of WatchTower is 15% smaller than *perfmon*.

3 Argus

The volume of WatchTower data and the rate at which it accumulates are daunting. Logging all counters on a typical machine at 1 Hz generates about 86.4 million values in a single day. And that is merely a single machine! Argus's approach to making sense of this plethora of data is to treat it as a dimensionality reduction (DR) problem, where each counter corresponds to a dimension. DR techniques, of which many exist in the statistics and machine learning literature [28], reduce high-dimensional data into a smaller number of dimensions, while retaining the majority of the information.

Consider a set of n performance counters (C_i) being logged every Δ seconds. A sample is the n-tuple $C_{j\Delta} = \{C_1(j\Delta), \ldots, C_i(j\Delta), \ldots, C_n(j\Delta)\}, i = 1, 2, \ldots, n, j = 1, 2, 3, \ldots$ a vector that reflects the n-element *state* of the machine at timestep j. The machine is also engaged in an *activity*, which is a high level description of how it is being used (e.g. web surfing, compute, backup, etc.) The size of a T second log is nT/Δ.

Given $\{C_0, C_\Delta, \ldots\}$, the following DR problems are posed to Argus:

1 What subset or function of the counters captures the interesting dynamics of the state? Such a subset consists of *subset counters*, and such a function produces a *composite counter*.

2 What is the high-level activity of the machine?

3 What is the appropriate Δ?

These problems are interdependent. Argus focuses on the first problem, and we have some preliminary results that address the second problem. The third is future work.

3.1 Data reduction techniques

To find compact state representations, Argus currently employs two DR techniques: principle components analysis and correlation elimination.

3.1.1 Principle Components Analysis (PCA)

Implicit in the original state representation (vector of counter values) is some n-dimensional orthonormal basis. Given this representation and samples of the state, PCA finds a new n-dimensional orthonormal basis such that the variance of the samples is concentrated on the new dimensions such that the first dimension captures as much variance as possible, the second captures as much of the rest as possible, and so on. The new basis consists of the eigenvectors of the covariance matrix of the data. Several methods for finding the basis exist, ranging in complexity from $O(mn^2 + n'n^2)$ to $O(n'mn)$ [15]. Basis-finding has also been parallelized [23]. Jain offers a concrete example of PCA from a performance analysis perspective [14, pp. 76–80].

Data reduction happens when we choose to use only the first n' of the new dimensions (our composite counters) as our representation. The value n' might be chosen so that the n' dimensions describe $> 90\%$ of the variance, for example. Each of the n' new composite counters (the principle components) is a simple weighted sum of the original counters—a projection onto the new dimensions. The weights associated with computing a principle component are referred to as its *loadings*.

For example, our application of PCA might find that `\Processor(0)\Priveledged Time` and `\Processor(0)\Interrupts/second` are correlated, while `Processor(0)\User Time` is negatively correlated with the first two. PCA would combine them into a single composite counter, weighting the first two counters positively and the third negatively.

3.1.2 Correlation Elimination

The correlation elimination (CE) algorithm is our method for selecting subset counters. Subset counters are preferable to composite counters when we must preserve the meanings of the original counters.

CE first computes the correlation matrix (correlations between all pairs of variables); and then finds the pair with the highest correlation, and throws out one of them. The assumption being that both are capturing similar information. This elimination of variables is repeated until a threshold correlation is reached. Applied to the previous example, CE might select `Processor(0)\Priveledged Time` and `\Processor(0)\%User Time` as the appropriate subset counters.

3.2 Evaluation

Our evaluation of Argus is based on applying the techniques described above to several large log files [24]. We applied PCA both within performance objects and across all performance counters to determine how many composite counters were needed to capture over 90% of the variance. We applied CE with each performance object to eliminate all correlations stronger than 0.85.

Activity	Number of Counters
Idle	45
Websurf	101
Compute	138
Overall	108

Figure 5: Number of counters with $c.o.v > 0.01$. The last row is computed for the entire trace, irrespective of activity.

3.2.1 Logs

Our evaluation is based on log files captured on four different machines using *perfmon*. This *Perfmon* data is equivalent to that captured by WatchTower. The reason we are using the *perfmon* data is that WatchTower was unavailable when we began this study.

The logs capture 244 different counters. We did not log transient performance objects such as processes, threads, and handles. We also ignored performance like Print Queue, Telephony, Indexing Service, Indexing Service Filter, etc which we believe to be non-critical/ inapplicable or were nearly constant. Each log is about three days long and is captured at a 1 Hz sampling rate.

We annotated the logs with the user activity being performed in each interval of time. The (*a priori* selected) activities are *idle* (the user was logged but no activity), *websurf* (1-8 windows open, and casual web-surfing), and *compute* (Matlab computation, Perl pre-processing of logfiles (disk-intensive), and occasionally running a Lisp interpreter). The annotations provide us with verification data. Argus does not use them to do data reduction.

In the following, we report on one of the traces, which was captured on dual-processor PIII 933 MHz machine with 512 MB of RAM.

3.2.2 Preprocessing

For some reason we do not understand, counter values are occasionally missing (5–10 per 100,000 samples) in the logs. These flaws appear to be randomly distributed and we do not consider them in our analyses.

We eliminated counters with coefficient of variation (cov) less than a minimum threshold (0.01). These counters did not change significantly during the trace or were constant parameters (such as total network bandwidth). Figure 5 shows the number of counters left after applying the cov threshold for data in the above three annotated activities as well as for all counters irrespective of activity.

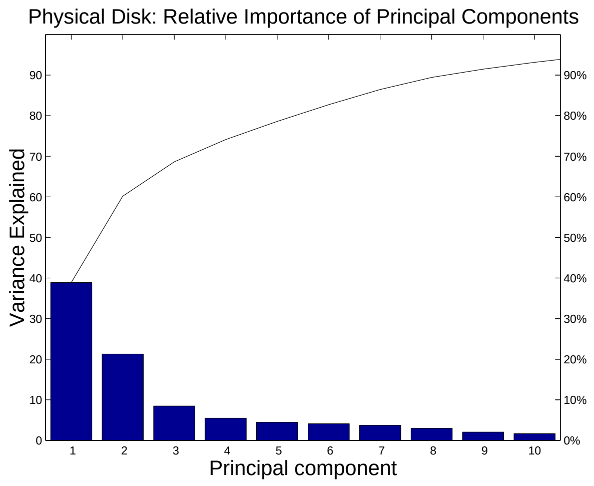

Figure 6: First PCs of Disk performance object. The bars show the loadings, and the thin line the cumulative variance explained.

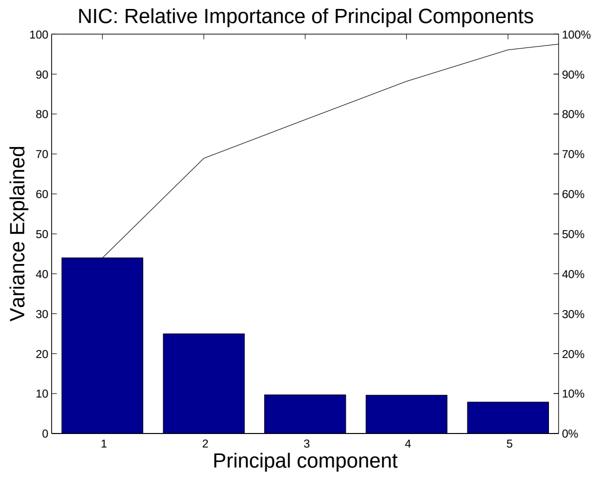

Figure 7: First PCs of Network performance object

3.2.3 PCA and CE within performance objects

To find out whether all of the counters within a performance object were actually needed, we performed PCA and CE for each performance object. As would be expected, there were large correlations within counters from the same performance objects, and this was observed by both methods. Figure 9 shows, for each performance object, how many composite counters (principle components) are needed to represent 90% of the variance in our log. The figure also shows the number of subset counters chosen by CE given that all correlations > 0.85 are eliminated.

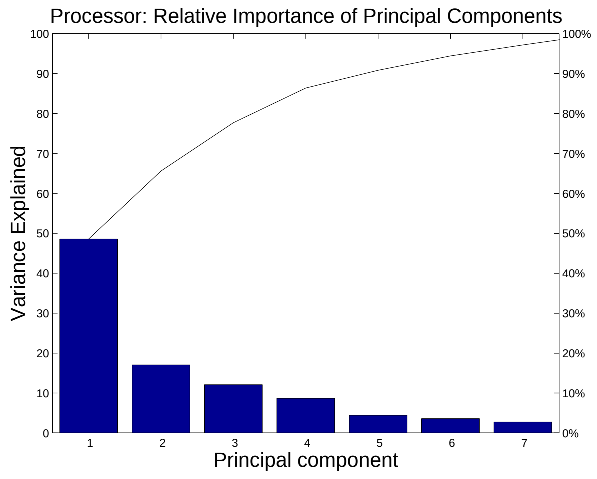

Figure 8: First PCs of Processor performance object

Performance Object	Total Ctrs.	PCA # Comps.	CE # Ctrs.
Cache	27	6	2
Memory	29	6	7
Network	34	4	2
IP	17	2	2
Objects	6	2	1
Disk	84	8	4
Processor	30	5	8
System	17	3	3
Total	244	36	29

Figure 9: Column 2 is the total number of performance counters in each object, column 3 is the number of composite counters (principle components) that are able to explain more than 90% of the variability in data, column 4 is the number of subset counters remaining after applying CE.

Figures 6, 7, and 8 show representative results for PCA on the disk, network, and processor performance objects. Each bar represents the contribution of a principle component (composite counter). In each case, only a few composite counters are needed to describe these objects.

3.2.4 PCA across all counters

The more interesting, but less a priori clear, correlations are those that exist across performance objects. We might expect that these correlations will depend upon the activity.

In the first analysis, PCA was carried out over the duration of the entire dataset, regardless of the activity. Figure 10 shows the contributions of the first 22 principle components of the entire dataset and how they capture more than 90% of the variability in the data.

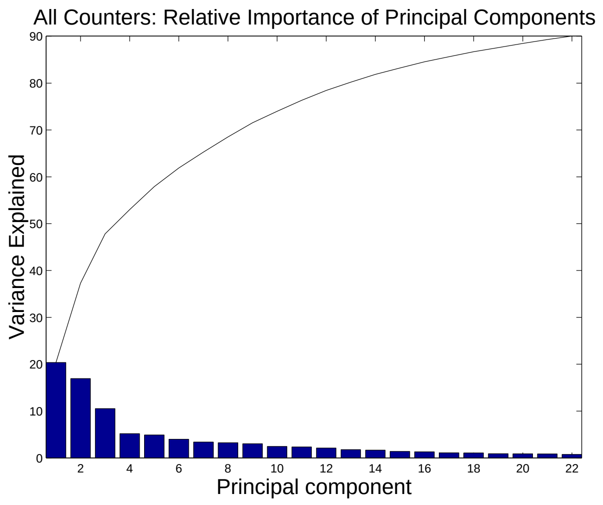

Figure 10: The first 22 principle components over the entire dataset.

Figure 11 plots the first two components as they evolve over time. The graph is annotated with activities. We can see clearly that the visual appearance of the plots is quite distinct in different activities. On datasets from two additional machines, qualitatively similar plots were observed.

We also did all-counter PCAs for each of the activities individually to see if there was a lot of difference in the number of principle components required to capture similar amounts of variability during the different activities. The general conclusion, over all the traces, is that during any activity we considered, the first 25 principle components (i.e., 25 composite counters) explain more than 90% of variability in the data.

Figure 12 shows the weights of the underlying counters that contribute to the counters of the first principle component for each of the different activities. Counters with weights close to zero could conceivably be ignored in computing the principle component, making this computation far cheaper.

The all-counter PCA (Figure 10) describes the state of the machine in 50% fewer composite counters (22 versus 36) than per-performance-object PCAs (total row of Figure 9). The tradeoff is that computing the single all-counter PCA is considerably more expensive than computing PCAs for each of the performance objects because the complexity of PCA grows at least quadratically with the number of original counters (5368 coefficients as opposed to 1391 with the per-performance-object PCAs). This result also raises an interesting possibility: perhaps

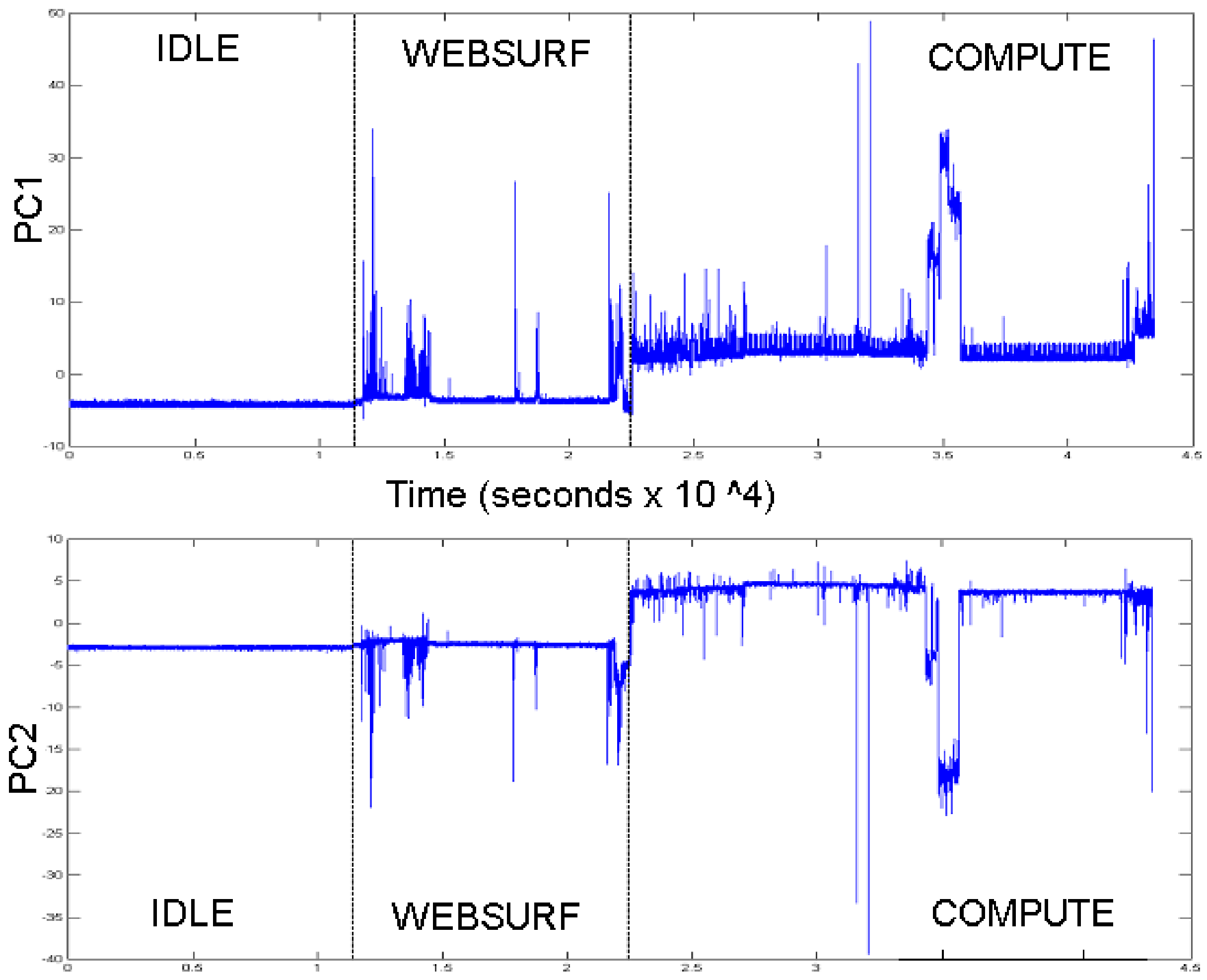

Figure 11: Plot of the first and the second principle component as the system goes through the three different states of idle, websurf and compute. This is a visual argument, but the statistical properties (like mean and variance) are clearly quite different allowing us to distinguish between different states

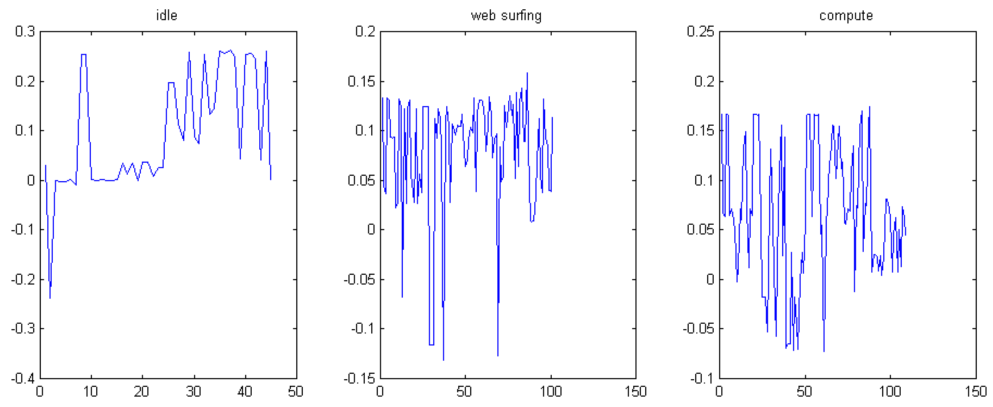

Figure 12: Loadings of the principle component in different states

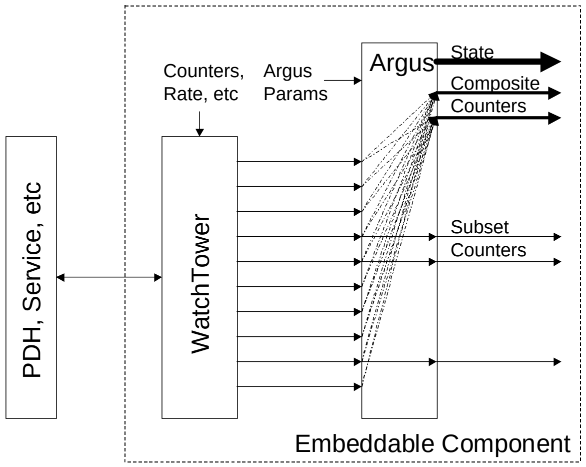

Figure 13: Integration plan for WatchTower and Argus.

data reduction techniques could be used to guide the construction of performance objects that have both a clear meaning to the user and high information content.

3.3 Overhead

As we do not yet have an online implementation of Argus, we don't know whether online data reduction will feasible. However, there is good evidence that it will be. The PCA described in the previous section was run on 3 days of 1 Hz data in about 10 minutes, which amounts to less than 3 ms of CPU time per sample. Thus we believe that doing the PCA online in a streaming fashion will require less than 1% CPU utilization. The CE was even cheaper. However, we are still working on how appropriately to do streaming PCA and CE. Computing correlation matrices incrementally is clearly easy, and CE's counter selection step is cheap, but computing eigenvectors incrementally may be expensive. It may be necessary to do PCA only occasionally.

4 Integration and future work

At this point, WatchTower and Argus are separate tools. WatchTower provides a simple, embedded interface to the Windows performance counters. Argus is a methodology for offline data reduction of Windows performance counter data that is implemented using interactive tools such as Matlab and S-Plus.

Our immediate plans are to build an online implementation of Argus and to integrate it with WatchTower to produce an embedded component, as shown in Figure 13. Given a list of counters and a sampling rate, WatchTower will provide streams of performance counter measurements to Argus. Given operating parameters (percentage of variation to capture, for example) Argus will reduce this data to composite counters using PCA and subset counters using CE. We also hope to add reduction to an activity. The WatchTower/Argus combination will continue to be embedded in other tools, such as RPS.

We plan to expand our Argus study to a larger and more diverse set of traces. WatchTower is currently running in our teaching lab, collecting traces of machine behavior. We also plan to use these traces to study the dynamic resource demands of interactive applications. We will consider more complex data reduction algorithms as needed.

5 Related Work

Performance monitoring and prediction systems such as Remos [19], NWS [43], and RPS [6, 7] have limited or nonexistent Windows support. This is not because the code is difficult to port, but rather because of the complex nature of sensors on Windows and the lack of the ability to embed *perfmon*-like tools. WatchTower provides a simple interface to Windows sensors and can be embedded into these systems.

Several Windows-specific monitoring systems do exist. Closest to our work is HPVM's monitor [32], which is explicitly targeted at Windows clusters. Unlike WatchTower, the HPVM monitor provides a complete monitoring infrastructure including communication and high precision clocks. In contrast, WatchTower provides a simple sensor service that can be embedded into other tools. WinInternals Software [42] and its SysInternals freeware website [35] provide interactive tools, such as Filemon and Regmon, that are specific for particular performance counters. NSClient [30] exposes Windows NT performance counters to the NetSaint monitoring system. None of these tools include Argus-like data reduction to capture the important dynamics in the data.

Data reduction is a large and complex field that is an outgrowth of multivariate statistics [4] and multivariate data analysis [10]. Recently, significant progress has been made in applying nonlinear methods to data reduction [29, 37]. Such methods may be appropriate for reduction of Windows data. The data reduction work that is closest to that in Argus is Vetter and Reed's application of dynamic statistical projection pursuit to the analysis of performance data from networked Unix systems [41].

The behavior of Windows systems and applications has begun to be studied. Chen, et al., compared the performance of different flavors of Windows and NetBSD on micro- and throughput-oriented application benchmarks using hardware cycle counters [1]. They followed up this work with a latency-oriented evaluation more suitable for interactive applications [9], and TIPME, a tool and framework for addressing latency problems in interactive applications [8]. Bershad, et al., have characterized the resource usage patterns of desktop and commercial applications running on Windows by instrumenting applications [16, 17]. Perl and Sites studied the performance of databases, compilers, and scientific codes on DEC Alpha machines running Windows NT using instruction and memory reference traces [25]. WatchTower may be helpful for studies such as these, providing a simple way to produce traces. Conversely, studies such of these can inform the development of Argus's data reduction algorithms.

6 Conclusions

We have described WatchTower, a system that simplifies the collection of Windows performance counter data, and Argus, a statistical methodology for making sense of this data and reducing the sheer volume of it. We are currently integrating WatchTower and Argus into a single embedded component that will provide easy access to the original performance counters, select statistically important subsets of the counters using correlation elimination, and provide new counters that compose information from multiple counters using principle components analysis.

We evaluated WatchTower and Argus with quite promising results. WatchTower provides easy access to raw performance counters with comparable overhead to *perfmon* running in non-GUI mode. Furthermore, WatchTower is easy to embed within other monitoring systems—we are currently building an RPS sensor based on it, for example. Argus can reduce the behavior of a large number of raw performance counters to few composite counters while still making it possible to determine the overall state of the machine and its interactive user. Argus can also trim the full set of raw counters to a reduced set of any size that contains the most significant uncorrelated counters.

References

[1] CHEN, J. B., ENDO, Y., CHAN, K., MAZIERIES, D., DIAS, A., SELZER, M., AND SMITH, M. D. The measured performance of pc operating systems. In *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP '95* (1995).

[2] CHIEN, A., LAURIA, M., PENNINGTON, R., SHOWERMAN, M., IANNELLO, G., BUCHANAN, M., CONNELLY, K., GIANNINI, L., KOENIG, G., KRISHNAMURTHY, S., LIU, Q., PAKIN, S., AND SAMPEMANE, G. Design and evaluation of an hpvm-based windows nt supercomputer. *International Journal of High-performance Computing Applications 13*, 3 (Fall 1999), 201–219.

[3] CONDOR TEAM. *Condor Version 6.2.0 Manual*. University of Wisconsin-Madison. Chapter 5, http://www.cs.wisc.edu/condor/manual.

[4] COOLEY, W. W., AND LOHNES, P. R. *Multivariate Data Analysis*. John Wiley and Sons, 1971.

[5] DINDA, P., LOWEKAMP, B., KALLIVOKAS, L., AND O'HALLARON, D. The case for prediction-based best-effort real-time systems. In *Proc. of the 7th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 1999)*, vol. 1586 of *Lecture Notes in Computer Science*. Springer-Verlag, San Juan, PR, 1999, pp. 309–318. Extended version as CMU Technical Report CMU-CS-TR-98-174.

[6] DINDA, P. A., AND O'HALLARON, D. R. An extensible toolkit for resource prediction in distributed systems. Tech. Rep. CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[7] DINDA, P. A., AND O'HALLARON, D. R. Host load prediction using linear models. *Cluster Computing 3*, 4 (2000).

[8] ENDO, Y., AND SELTZER, M. Improving interactive performance using tipme. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '00)* (2000), pp. 240–251.

[9] ENDO, Y., WANG, Z., CHEN, J. B., AND SELTZER, M. Using latency to evaluate interactive system performance. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation* (1996).

[10] FLURY, B., AND RIEDWYL, H. *Multivariate Statistics: A Practical Approach*. Chapman and Hall, 1988.

[11] GALSTAD, E. http://netsaint.sourceforge.net/, 2001.

[12] GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHECK, R., AND SUNDERAM, V. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, Massachusetts, 1994.

[13] HOFMEYR, S. A., FORREST, S., AND SOMAYAJI, A. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6 (1998), 151–180.

[14] JAIN, R. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[15] JOLLIFFE, I. T. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[16] LEE, D., CROWLEY, P., BAER, J.-L., ANDERSON, T., AND BERSHAD., B. Execution characteristics of desktop applications on windows nt. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA '98)* (June 1998).

[17] LEE, D., DAVIS, A., ULBRIGHT, C., AND BERSHAD, B. Characterizing commercial workloads under windows nt. In *Proceedings of the ASPLOS '98 Workshop on PC Performance Characterization* (1998).

[18] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor—a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computer Systems (ICDCS '88)* (June 1988), pp. 104–111.

[19] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 1998), IEEE, pp. 189–196.

[20] MICROSOFT CORPORATION. Microsoft platform software development kit. http://microsoft.com/windows2000/library/resources/reskit/.

[21] MICROSOFT CORPORATION. Windows 2000 professional resource kit. http://microsoft.com/windows2000/library/resources/reskit/, 2000.

[22] MICROSOFT CORPORATION. Monitoring performance. http://www.cwlp.com/samples/tour/perfmon.htm, 2001.

[23] MILOSAVLJEVIC, I. J., PARTRIDGE, M., CALVO, R. A., AND JABRI, M. A. High performance principal component analysis with paral. In *Proceedings of the Ninth Australian Conference on Neural Networks, Brisbane* (1998).

[24] PARITOSH, P. Windows monitoring logs. http://www.cs.nwu.edu/ paritosh/WinTraces/, 2001.

[25] PERL, S. E., AND SITES, R. L. Studies of windows nt performance using dynamic execution traces. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI '96)* (October 1996), pp. 169–184.

[26] PLATFORM COMPUTING CORPORATION. Load sharing facility. http://www.platform.com.

[27] RAJKUMAR, R., LEE, C., LEHOCZKY, J., AND SIEWIOREK, D. A resource allocation model for QoS management. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 1997).

[28] RENCHER, A. C. *Methods of Multivariate Analysis.* Wiley, New York, 1995.

[29] ROWEIS, S. T., AND SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 (2000), 2323–02326.

[30] RUBIN, Y. Nsclient. http://netsaint.sourceforge.net.

[31] SAAVEDRA-BARRERA, R. H., SMITH, A. J., AND MIYA, E. Machine characterization based on an abstract high-level language machine. *IEEE transactions on computers 38*, 12 (December 1989).

[32] SAMPEMANE, G., PALKIN, S., AND CHIEN, A. Performance monitoring on an hpvm cluster. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '00)* (June 2000).

[33] SCHOPF, J. M., AND BERMAN, F. Stochastic scheduling. In *Proceedings of Supercomputing '99* (1999). Also available as Northwestern University Computer Science Department Technical Report CS-99-03.

[34] SNAVELY, A. Performance modeling and prediction for the hpc community. http://www.sdsc.edu/ allans/position.html, 2001.

[35] SYSINTERNALS LLC. http://www.sysinternals.com/.

[36] TAKEDA, K., ALLSOPP, N. K., HARDWICK, J. C., MACEY, P. C., NICOLE, D. A., COX, S. J., AND LANCASTER, D. J. An assessment of mpi environments for windows nt. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)* (1999).

[37] TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290 (2000), 3219–2323.

[38] THOMPSON, N. Creating a simple win32 service in c++. Microsoft Windows Platform Technical Article, November 1995.

[39] TOMLINSON, P. *Windows NT Programming in Practice: Practical Techniques from Master Programmers.* R&D Books, 1997.

[40] VETTER, J., AND MUELLER, F. Profiles in courage: Explicit communication characteristics of scientific applications. In *Submitted to SC'01* (2001).

[41] VETTER, J. S., AND REED, D. A. Managing performance analysis with dynamic statistical projection pursuit. In *Proceedings of Supercomputing '99 (SC '99)* (November 1999).

[42] WINTERNALS SOFTWARE LP. http://www.winternals.com/.

[43] WOLSKI, R. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)* (August 1997), pp. 316–325. extended version available as UCSD Technical Report TR-CS96-494.

[44] WOODWARD, P. Personal communication, April 2001.