The Next Frontier: Interactive and Closed Loop Performance Steering*

Daniel A. Reed
Christopher L. Elford
Tara M. Madhyastha
Evgenia Smirni
Stephen E. Lamm

{reed,elford,tara,esmirni,slamm}@cs.uiuc.edu
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Abstract

Software for a growing number of problem domains has complex, time varying behavior and unpredictable resource demands (e.g., WWW servers and parallel input/output systems). While current performance analysis tools provide insights into application dynamics and the causes of poor performance, with a posteriori *analysis one cannot adapt to temporally varying application resource demands and system responses. We believe that the solution to this performance optimization conundrum is integration of dynamic performance instrumentation and on-the-fly performance data reduction with real-time adaptive control mechanisms that select and configure resource management algorithms automatically, based on observed application behavior, or interactively, through high-modality virtual environments. We motivate this belief by first describing our experiences with performance analysis tools, input/output characterization, and WWW server analysis, and then sketching the design of interactive and closed loop adaptive control systems.*

1 Introduction

To attack problems of increasing complexity, application developers are eschewing regular, easily parallelized algorithms in favor of more complicated but flexible approaches that can exploit the multiplicity of temporal and spatial scales inherent in multidisciplinary problems. In consequence, an increasing fraction of important applications have complex, data dependent execution behavior and time varying resource demands. Because the interactions between application and system software change during and across application executions, it is difficult or impossible to determine a globally optimal application configuration or to statically configure runtime systems and resource management policies.

As the scope of scalable parallel computing expands from regular computations and single parallel systems to irregular computations and distributed collections of heterogeneous parallel systems, software complexity and associated optimization problems increase commensurately. This view is buttressed by our experiences characterizing parallel input/output behavior [2] and analyzing WWW server traffic patterns [7]. Applications in both domains have complex, irregular resource access patterns that make static allocation of resources inefficient.

In contrast to rapid changes in application software models, tools for debugging and tuning parallel software have not kept pace. Simply put, current software tools are data limited and enforce a passive view — users and developers must manipulate text and graphics via a small workstation window on the complex world of large-scale, heterogeneous metacomputing, with few options to adapt application behavior or resource management policies during execution.

We believe the solution to the performance optimization conundrum is integration of dynamic performance instrumentation and on-the-fly performance data reduction with configurable, malleable resource management algorithms, and real-time adaptive control mechanisms that either automatically choose and configure resource management algorithms based on application request patterns and observed system performance or allow users to interactively guide application behavior. To capitalize on human instincts and skills, interactive guiding is best done in high-modality virtual environments where users can directly manipulate software representations and behavior.

Building on this thesis, in §2, we describe the design of the Pablo performance analysis environment and the motivations for its current structure. In §3–§4, we describe our experiences using the Pablo software to capture and analyze the input/output dynamics of large, input/output intensive parallel applications and of large-scale WWW servers. Based on these ex-

*This work was supported in part by the Advanced Research Projects Agency under ARPA contracts DAVT63-91-C-0029 and DABT63-93-C-0040, by the National Science Foundation under grants NSF IRI 92-12976, NSF ASC 92-12369, and NSF CDA 94-01124, and by the National Aeronautics and Space Administration under NASA Contracts NGT-51023, NAG-1-613, and USRA 5555-22.

periences, in §5 we describe the lessons learned from this analysis, and in §6–§7 sketch a vision of open and closed loop adaptive control systems for resource management policy selection. In §8 we describe related work. Finally, §9 summarizes the current state of our work and outlines plans for continued research.

2 Performance Analysis Software

The critical importance of robust, flexible, and efficient performance tools has long been recognized and has been a major discussion topic at several national meetings [9, 15]. By necessity, such tools must embody knowledge of the execution environment and identify performance bottlenecks in terms of application code interaction with the execution environment. Moreover, because the root causes of poor performance or unexpected program behavior may lie with run-time libraries, compilers, operating system, or hardware, tools must gather and correlate information from many sources.

Ideally, a complete performance analysis environment should include instrumentation at multiple hardware and software levels, real-time and post-mortem data reduction, and an extensible set of data analysis and correlation tools. The environment also should be *portable* across a range of parallel and distributed systems, allowing users to amortize learning costs across many contexts, *scalable* with the size of the system being studied, allowing performance optimization on large systems, and *extensible*, allowing addition of functionality as needed. These beliefs are based on our experiences building three generations of hardware and software performance analysis tools for parallel systems [8, 13], as well as the collective experiences of the performance tool community [15].

For the past five years, we have worked to develop a software performance analysis infrastructure, called Pablo [11], that embodies the design goals of portability, extensibility, and scalability. Below, we describe the components of the Pablo environment, the keys to its adaptability, and examples of its application to selected problem domains.

2.1 Pablo Performance Environment

As we described in [12], the complete Pablo[1] performance environment consists of the following:

- *An extensible self-defining data format* (SDDF) and associated library that separates the structure of performance data records from their semantics, allowing easy addition of new performance data types.
- *An instrumenting parser* capable of generating instrumented SPMD source code.
- *Extensible instrumentation libraries* that can capture event traces, counts, or interval times and reduce the captured performance data on the fly.
- *Graphical performance data display and sonification tools*, based on coarse-grain graphical programming, that support rapid prototyping of performance analyses.

[1] Pablo is a registered trademark of the Board of Trustees of the University of Illinois.

The Pablo instrumentation software captures dynamic performance data via instrumentation library calls that are inserted in the application source code. Typically, these library calls bracket key software constructs (e.g., procedure, input/output, or communication calls) and invoke the instrumentation library to record timestamps, durations, and parameters of the constructs. During program execution, the performance data that can be either directly recorded by the data capture library or processed by one or more data analysis extensions prior to recording (e.g., to generate a statistical profile from a timestamped event stream).

As the performance data is generated and optionally processed, it can be extracted via network sockets for real-time display, or with the requisite interfaces, used to interactively control application or system behavior; see §7. Alternatively, after program execution completes, the data can be analyzed by a toolkit of data transformation modules capable of processing the self-defining data format (SDDF). Just as the data flow prototyping model proved successful in visualization (e.g., with AVS and SGI Explorer), the Pablo coarse-grain, graphical data flow support for rapid prototyping of performance analyses makes it possible to quickly construct new data analyses and graphical displays.

2.2 Pablo Software Adaptability

To satisfy the design goals of portability and extensibility, the Pablo environment separates performance data structure and data semantics via the self-defining data format (SDDF) and isolates the implementation details of performance data capture on a particular system behind a set of standard library interfaces. The former has allowed us to easily define and process new types of performance data without changing the underlying Pablo software, and the latter has made possible broad extension of the data capture library.

Although several data meta-formats have been developed (e.g., netCDF and HDF) for grid-based scientific data, SDDF was designed and optimized specifically to represent record-based performance data. SDDF data streams consist of an initial group of record descriptors and a subsequent stream of record instances. The SDDF descriptors define the structure of the record instances in the same way as record declarations in Pascal or C, though SDDF records are limited to scalars and arrays of character strings, integers, and single and double precision floating point values. The data following the SDDF descriptors consists of a stream of descriptor tag and data record pairs. Each tag identifies the record descriptor that defines the immediately following data.

By separating the structure of data from its semantics and allowing one to define new performance data records appropriate to a particular instrumentation context, the SDDF library allows one to construct tools that can extract and process SDDF records and record fields with minimal knowledge of the data's deeper semantics. Moreover, the Pablo coarse-grain data flow model for rapid prototyping embodies performance analysis semantics in the analysis graph construction, where one interactively specifies what

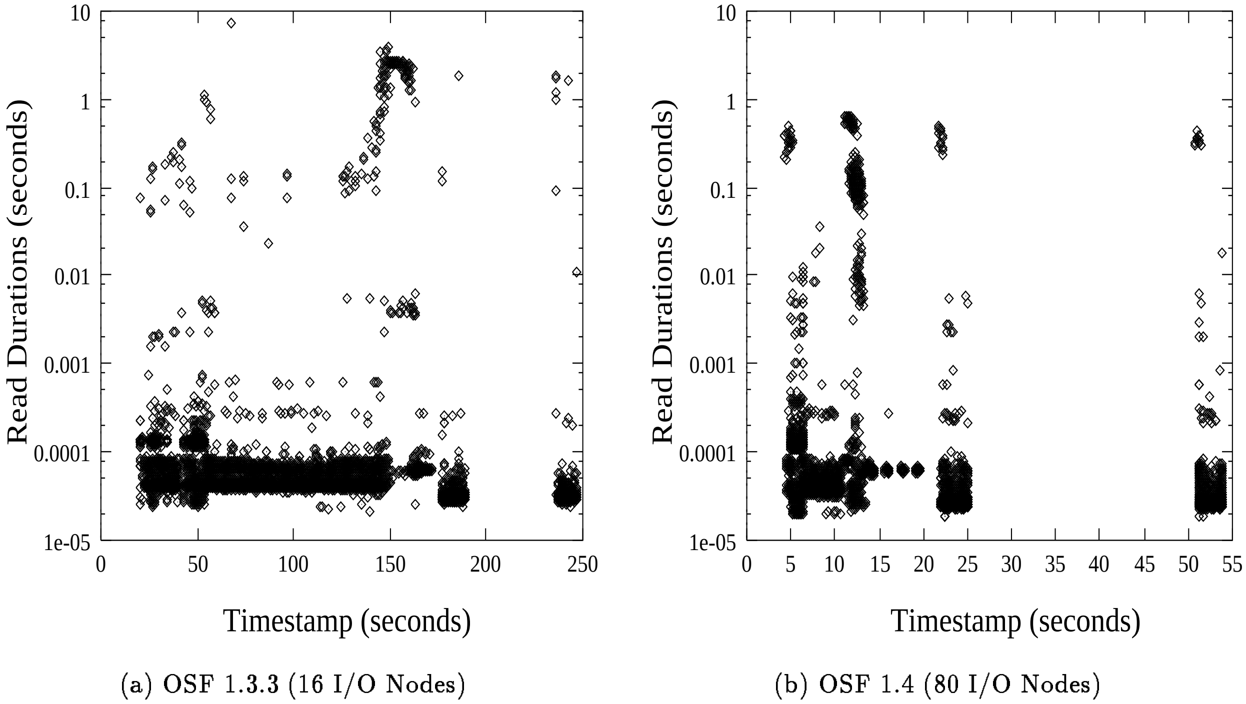


Figure 1: File Read Durations During Execution (PRISM on Intel Paragon XP/S)

SDDF records and fields should be processed, rather than in the semantics of the individual graph modules.

The second key to Pablo's adaptability is the design of a set of software extension interfaces for the Pablo data capture library. By separating low-level instrumentation details (e.g., timestamp generation and data extraction) from data processing, one can develop platform-independent data analysis extensions that process SDDF performance data records prior to extraction. Exploiting the Pablo instrumentation library's extension interfaces, we have expanded the instrumentation library beyond its initial target of message passing codes to include support for analysis of codes written in data parallel languages [1], study of World Wide Web behavior [7], analysis of application input/output patterns [2, 16], and study of parallel file system policies [4].

3 Input/Output Characterization

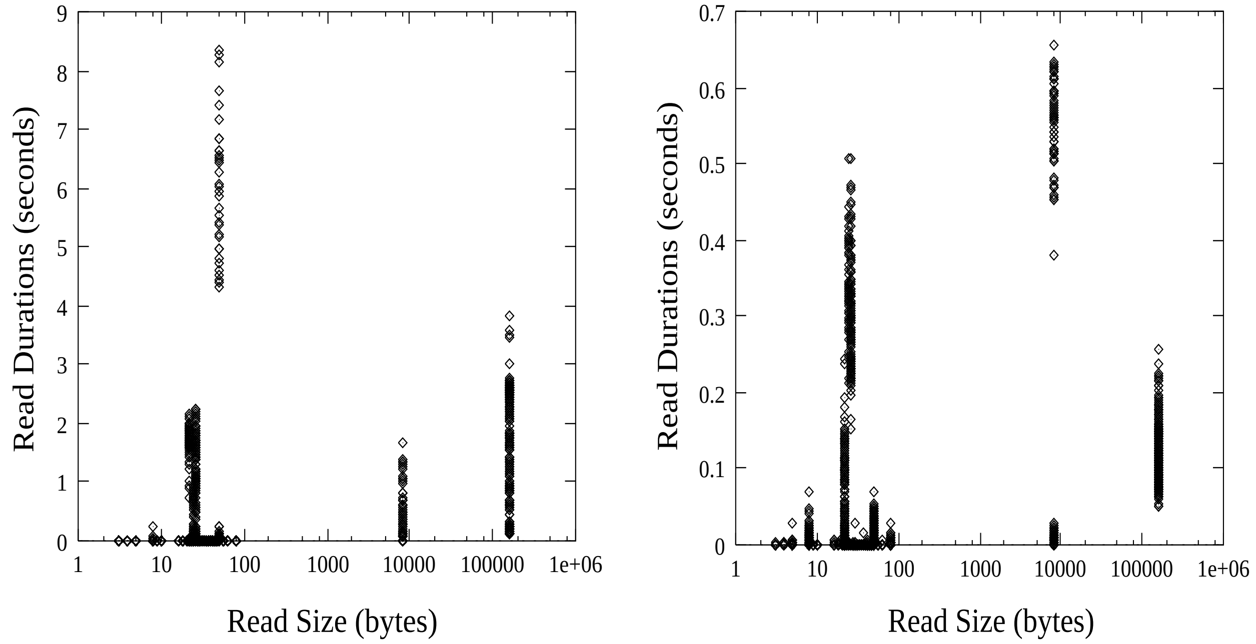
The modest input/output configurations and file system limitations of many current high-performance systems preclude solution of problems with large input/output needs. Moreover, technology trends are increasing the already wide disparity between the performance of secondary and tertiary storage devices and the performance of processors and communication links. In consequence, input/output hardware and file system parallelism is the key to bridging the performance gap. This requires detailed knowledge of the input/output characteristics of scalable parallel applications and exploitation of such knowledge to design and manage parallel input/output systems. In short, one must characterize extant input/output patterns both quantitatively (i.e., request sizes and durations) and qualitatively (i.e., why applications have particular access patterns).

3.1 Characterization Techniques

As part of the national Scalable I/O (SIO) initiative [10], we have extended the Pablo infrastructure to capture and analyze the temporal and spatial patterns of parallel application input/output. The initial focus on application level characterization was motivated by a desire to isolate and identify file access patterns that were intrinsic to each application and to understand the interactions between application request patterns and the hardware and software of parallel input/output systems.[2]

Using the Pablo instrumentation library extension mechanisms, the input/output characterization software supports either detailed tracing of individual input/output calls or statistical summarization of input/output activity. The former generates event trace records that include the time, duration, size, and other parameters of each input/output operation. When detailed event trace capture might generate an excessive volume of data, perturbing the input/output system being measured, statistical analysis can reduce the trace data before extraction. These statistical sum-

[2] At present, we are augmenting application data with instrumentation of input/output device drivers to capture physical input/output data.



(a) OSF 1.3.3 (16 I/O Nodes) (b) OSF 1.4 (80 I/O Nodes)

Figure 2: File Read Durations and Request Sizes (PRISM on Intel Paragon XP/S)

maries can take one of three forms: file lifetime, time window, or file region. The first describes the sizes and latencies of requests during the time interval that input/output occurs to a given file. The second and third are spatial and temporal analogs, describing access patterns during an interval of time or a file region, respectively.

3.2 Input/Output Evolution

During the past three years, we and others have exploited the extended Pablo analysis software to study the behavior of a variety of parallel applications on the Intel Paragon XP/S and IBM SP/2 [2, 16]. These studies have shown that parallel input/output patterns are much more complex and irregular than those expected based on an extrapolation from input/output patterns on either vector supercomputers or high-performance workstations.

As one example, we have tracked the evolving performance of a computational fluid dynamics code (PRISM) across two releases of the Intel Paragon OSF/1 parallel file system (PFS) and two disk hardware configurations (sixteen and eighty I/O nodes). PRISM is a parallel implementation of a 3-dimensional numerical simulation of the Navier-Stokes equations. This particular code models a geometry where the flow is periodic in at least one direction (e.g., flow past a cylinder, flow in a channel, or flow over a backward-facing step). An initial velocity field is given by the input data, and the solution is integrated forward in time by solving the equations that describe advection and diffusion of momentum in the fluid.

Figure 1 shows the behavior of one snapshot of the PRISM code that uses Unix input/output primitives on the Intel Paragon XP/S. In Figure 1a, the code executed on Intel OSF version 1.3.3 with all application files striped across sixteen I/O nodes, and in Figure 1b, the code executed on Intel OSF version 1.4 with files striped across eighty I/O nodes. Not only does the total execution time differ by a factor of five between Figures 1a and 1b, but the temporal patterns of file reads and their individual read durations change dramatically. Comparison of Figures 1 and 2 shows the temporal compression of read requests in time and as a function of request size.

Although operating system optimizations are important, additional experiments with OSF 1.4, but using only sixteen I/O nodes, showed that the larger input/output hardware configuration is the dominant cause of higher input/output performance. To understand the relation between file system semantics, input/output hardware configurations, and application request patterns, we have also measured the performance of two other versions of the PRISM code that exploit the Intel PFS primitives. We found that performance differences among the three versions arose only when the number of I/O nodes was samll, limiting input/output parallelism. This suggests that input/output optimization strategies are strongly sensitive to the hardware configuration and the access pattern.

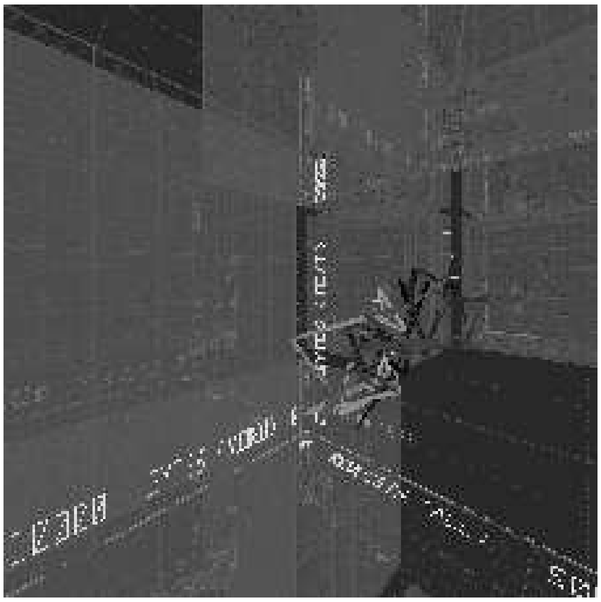


Figure 3: *Avatar* Scattercube WWW Display

3.3 Input/Output Observations

The complex behavior shown in Figures 1 and 2 is typical of that observed in parallel applications on both the Intel Paragon XP/S and the IBM SP/2. Not only do parallel input/output requests vary widely in size, with both very small (i.e., tens of bytes) and very large (i.e., tens of megabytes) requests, but input/output concurrency varies widely as well, ranging from single processor input/output to concurrent access by all processors. Likewise, request patterns can be sequential, strided, or random with fixed or variable sizes.

Overall, our characterization studies [2, 16] have shown that parallel applications exhibit a much wider variety of input/output request patterns than anticipated by parallel file system designers. Thus, extant parallel file systems are optimized for a small subset of the parallel input/output access space. For example, large requests are best managed by streaming data directly to or from storage devices and application buffers. Conversely, small requests are better served by prefetching, caching, and write-behind.

Given the diversity of parallel input/output patterns, we believe parallel file systems that rely on a single, system-imposed policy are unlikely to be successful. Instead, exploitation of input/output access pattern knowledge in caching and prefetching systems is crucial to obtaining a substantial fraction of peak input/output performance. Inherent in such an adaptive approach is the need to identify access patterns and choose policies based on access pattern characteristics. We will return to this notion of dynamic adaptation based on changing application resource demands in §6–§7.

4 WWW Characterization

The World Wide Web (WWW), together with rapidly expanding interests in distributed data mining, poses a decidedly different, though equally thorny, set of performance optimization problems than those faced when optimizing input/output behavior on a single parallel system. In particular, WWW servers now place demands on operating systems, input/output systems, and network protocol implementations that lie far outside the system software's original design points. Moreover, as the WWW is increasingly used as a geographically distributed multimedia data base, wide area distributed query optimization must exploit both query semantics and time-varying data on the computation and input/output capabilities of heterogeneous WWW servers connected by networks of widely varying bandwidth and latency. Thus, understanding extant and evolving request patterns and the loads placed on both servers and networks is key to optimizing both server design and operating system resource management policies.

4.1 Characterization Techniques

Due to its early development of the Mosaic WWW browser and its role as an NSF supercomputer center, the National Center for Supercomputing Applications (NCSA) processes several hundred thousand WWW requests each day. To understand the access patterns to this server and the server responses, NCSA logs not only the request stream but also samples of the server network statistics, disk input/output, and processor utilization [6]. Given the diversity of request patterns and the size of the NCSA WWW site, these logs exceed 150 MB/day and have been archived for over two years. Although the NCSA WWW logs provide the data needed to understand both short and long term evolution of WWW traffic, their sheer size makes data analysis challenging.

We extended the Pablo instrumentation and analysis software to process process the NCSA logs, creating a set of SDDF records that describe access pattern stimuli and server responses. Collectively, these SDDF records define nearly forty access pattern and server metrics that are computed each minute. To understand and correlate these metrics, we exploited Pablo's support for real-time data analysis and extraction to drive an immersive virtual environment, called *Avatar* [12, 14, 7], that supports real-time, three-dimensional visual and aural display of dynamic performance data.

Figure 3 shows a three-dimensional scatterplot, one of several *Avatar* virtual environment displays. In Figure 3, the three axes of the cube correspond to three different performance metrics, with the colored ribbons showing the trajectory of the WWW servers within the metric space. By interactively changing the metrics mapped to each axis, one can explore the correlation of access patterns and server responses.

4.2 WWW Observations

As we described in [6], analyzing several months of accesses to the NCSA WWW server showed that commercial and government use of the server is growing rapidly, and request heterogeneity is increasing, lessening server cache hit ratios. In addition, the data clearly show that increasing use of non-text data (i.e., scientific data sets, imagery, audio, and video) will require different approaches to WWW server design. For example, data cache partitioning would ameliorate the pernicious effects of retrieving large images by reserving a portion of the cache for smaller, text items. Likewise, streaming video techniques that guarantee quality of service without preloading an entire video segment would lessen memory requirements. In general, WWW servers must become more adaptive and intelligent, with flexible file caching and prefetching strategies that can adapt to changing request patterns and client types.

5 Performance Analysis Lessons

The common theme underlying the input/output characterization and WWW traffic analysis in §3 and §4 is widely varying access patterns and concomitant resource demand variability. In both cases, experience suggests that optimizing performance will require a judicious match of resource management policies to resource request patterns. Parallel file systems should dynamically configure caching and prefetching policies, as well as cache sizes, based on access pattern attributes observed during program execution. Likewise, sophisticated WWW servers should exploit knowledge of request data types, client capabilities, network connection bandwidths, and user-specific request patterns.

Although the current Pablo software infrastructure's flexibility allowed us to quickly construct the performance analysis and display tools needed to understand the rapidly varying resource demands of input/output intensive applications and WWW servers, it is not sufficient. Because the interactions between dynamic, irregular applications and system software change during application execution, we believe that a flexible infrastructure for performance directed adaptive control must replace *a posteriori* performance analysis. Such an infrastructure would allow application and system developers to create adaptive software that can change its behavior in response to real-time varying resource demands.

In §6–§7, we describe two complementary views of such an adaptive control infrastructure. The first, a closed loop design shown in Figure 4, relies on real-time performance data captured using a group of distributed performance sensors, a suite of distributed decision procedures to select resource management policies, and a complementary set of policy actuators to realize policy decisions. The second, an interactive design shown in Figure 7, replaces software-directed resource policy selection with immersive environments for real-time performance display and user-directed resource management.

6 Closed Loop Adaptive Control

Minimally, an infrastructure capable of dynamically adapting to changing resource demands would (a) monitor resource demands and system responses, (b) select resource management policies based on monitored data, and (c) implement policy changes in globally consistent ways. A more general adaptive control system might contain additional components for capture of qualitative as well as quantitative performance data, off-line assessment of policy effectiveness, and flexible configuration of decision procedures.

Creating a flexible closed loop control system that could be used on multiple parallel and distributed systems to manage a variety of physical and logical resources is a daunting task that poses a plethora of open research problems. Below, we first sketch one possible design and briefly describe our current research within the context of this design.

6.1 Proposed Closed Loop Design

Figure 4 shows one possible design for a general-purpose, closed loop resource management and control system containing the following components:

- *Distributed performance sensors* that can capture quantitative application and system performance data and compute performance metrics.
- *Software actuators* that can enable and configure resource management policies.
- *Program assertion interfaces* for qualitatively specifying application resource request patterns (e.g., large read requests or short computation intervals) and *automatic behavioral classification techniques* for use when assertions are missing or inaccurate.
- *Decision procedures*, both local (e.g., per parallel task) and global (e.g., per parallel program), for selecting resource management policies and enabling actuators based on observed application resource requests and the system responses captured by performance sensors.
- *Off-line analysis tools* to assess the performance of current decision procedures, to conduct parametric performance studies using captured application behavioral data, and to identify improved decision procedures.

In this design, performance instrumentation sensors capture and compute quantitative application and system performance metrics. Qualitative application behavior is obtained from user assertions or via automatic behavioral classification techniques. Together, the qualitative and quantitative data are used by a hierarchy of decision procedures to choose and configure resource management policies via software actuators. Finally, off-line analysis tools permit meta-performance assessment of the adaptive control system's efficacy.

6.2 Control Prototype Experiences

To test the feasibility of the design just described, we have enhanced our user-level portable parallel file system (PPFS) [4] to include prototype sensors, actuators, decision procedures, and behavioral classification software. PPFS consists of a group of cooperating file servers and has a rich application interface, allowing applications to advertise access patterns and to control caching, prefetching, and data placement policies at multiple levels. The PPFS prototype relies on the Pablo library to capture and compute quantitative performance metrics, includes an artificial neural

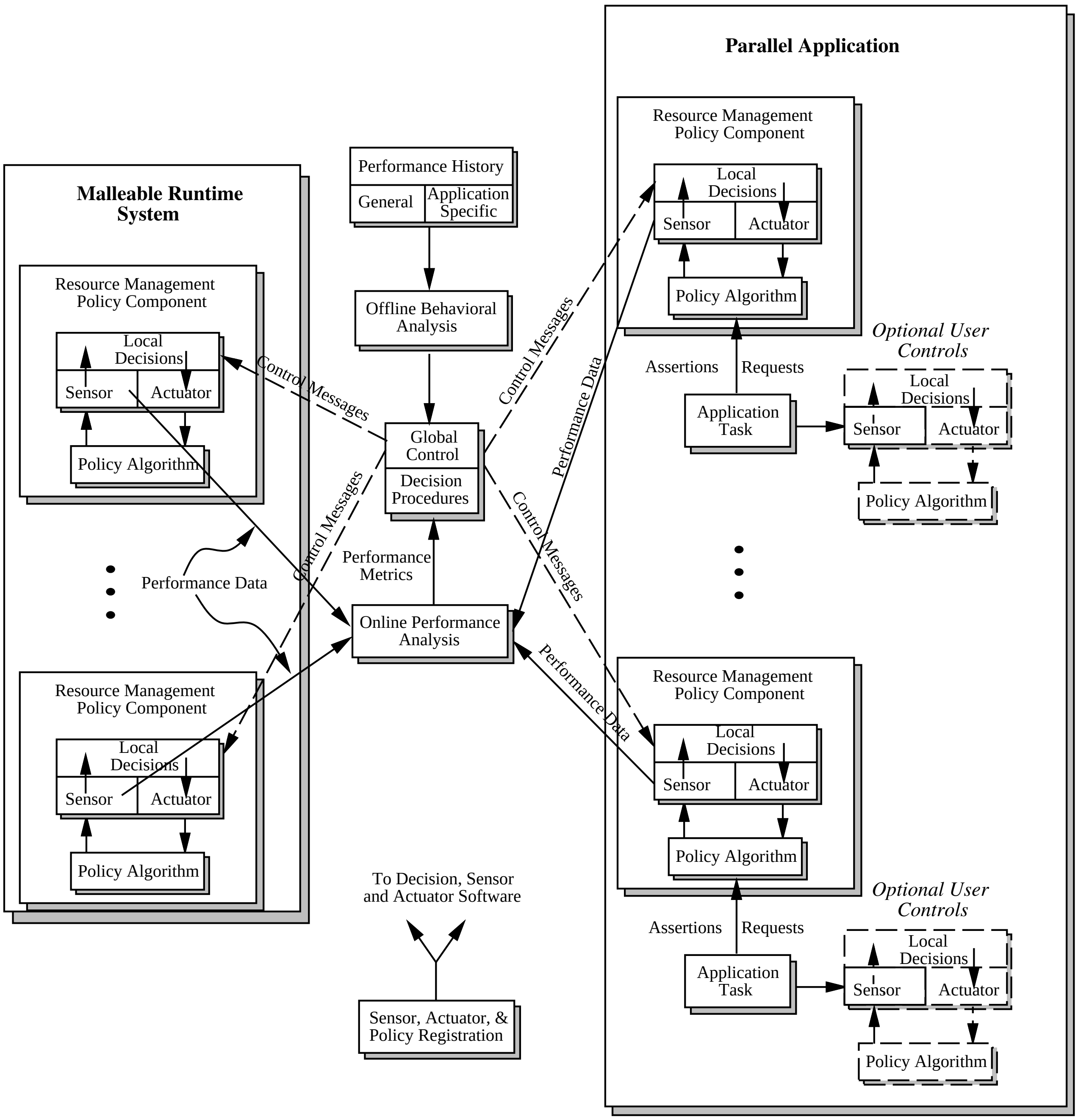


Figure 4: Closed Loop Resource Management Control Infrastructure

network that accepts file access statistics and generates qualitative access patterns, and relies on a simple set of decision procedures to select file policies.

The goal of the PPFS design is to improve parallel input/output performance by adaptively selecting and tuning file system policies based on application request patterns. Below, we describe qualitative and quantitative behavioral classification techniques and our preliminary experiences using these approaches to tune PPFS policies. Ultimately, we believe that combining the qualitative and quantitative approaches to policy selection will create a rich environment for dynamic, closed loop policy steering within PPFS.

6.2.1 Automatic Qualitative Classification

Given a qualitative classification of behavior (.e.g., that file accesses are sequential and small), one can select a resource management policy family best matched to the behavior type. Such qualitative data can be obtained by inserting behavioral assertions in source code or by monitoring and automatically classifying behavior. Automatic classification lessens the application programming burden, by transparently identifying application access patterns.

To automatically classify file access patterns, we have identified a set of features commonly observed during our characterization of scientific applications. These features, shown in Table 1, partition requests along three axes: read/write mix, sequentiality, and size. Using this partitioning, we trained a feedforward artificial neural network to classify input/output requests during application execution. The output of the trained neural net corresponds directly to the features in Table 1. Not only is such a qualitative classification system independent, it may be unique to a particular application execution. By using the classification during execution to drive file policy actuators, one can transparently tune caching and prefetching policies for a given system and set of application inputs with application hints or optimizations.

As a validation of automatic behavioral classification and dynamic adaptation, we used the enhanced PPFS to improve the input/output performance of a a single processor application from the NOAA/NASA Pathfinder AVHRR (Advanced Very High Resolution Radiometer) data processing project. Pathfinder processing is typical of low-level satellite data processing applications — fourteen large files of AVHRR orbital data are processed to produce a large output data set.

For the Pathfinder code, the automatic behavioral classification and adaptive file system policies of PPFS offer a significant performance improvement over that possible with UNIX buffered input/output. The neural net access pattern classifier detects that the output file access pattern is initially write only and sequential, with large accesses; later, the access pattern changes to write only and strided, with very small accesses. PPFS chooses an MRU cache block replacement policy for the first phase. In the second phase, it enlarges the cache to retain the working set of file blocks.

Figure 5 shows the file write durations during the first phase of Pathfinder execution, both with and without the PPFS adaptive policies, with executed on a single processor SPARC server. The first cluster of accesses at the left of Figures 5a and 5b is the sequential write phase. Performance for the first phase is roughly equivalent using either MRU or the default, non-adaptive LRU replacement policy. However, enlarging the cache in the second, strided access phase substantially decreases the average write duration and overall execution time.

In general, we have found that automatic file access pattern classification and policy selection yields major performance benefits with modest overhead. Currently, we are extending this effort with a more general classification infrastructure that supports global access pattern classification across multiple processors.

6.2.2 Sensor Quantitative Classification

Qualitative access pattern classification, used to select and tune resource management policies, is a primary step toward creation of adaptive, closed loop control. However, it must be coupled with quantitative data on resource utilization to provide true closed loop control. In a complementary effort to explore the efficacy of adaptive control, we have coupled performance sensor data with user assertions that describe application input/output patterns.

To automatically determine the input/output performance of the file system, we have instrumented PPFS with the Pablo instrumentation library. This instrumented PPFS produces sensor data as sliding window averages of key performance metrics (e.g., file read byte counts and durations, file cache hit ratios, input/output server timings, queue lengths and delays, and prefetch initiation overheads).

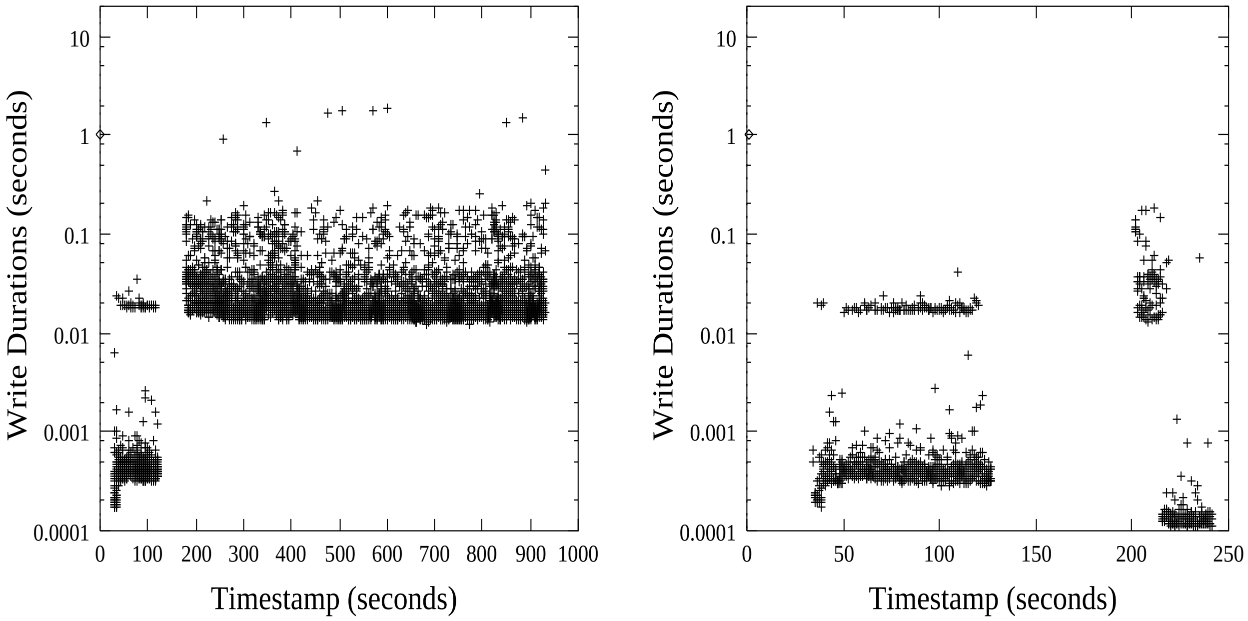
Together with application assertions about qualitative access patterns, the PPFS sensor data can be processed in real time by an actuator library to identify and eliminate the current PPFS bottleneck. For example, if the file access pattern is sequential and the file interaccess time declines, the cache hit ratio sensor may decline as well, indicating that PPFS should increase the prefetch parameter and more aggressively prefetch data to increase the cache hit ratio. Because the effects of such changes are manifest as changes in the sensor data, the control infrastructure can quickly determine the veracity of the control change.

To demonstrate the efficacy of sensor-based adaptive control when coupled with behavioral assertions, we used an input/output benchmark to conduct a set of simple experiments on the IBM SP/2 and on two different hardware configurations of the Intel Paragon XP/S. In this benchmark, all processors read disjoint, interleaved 4 KB blocks of the same file, forming a globally sequential access stream. Using the enhanced PPFS, we measured the average read access time as a function of file interaccess latency and file cache prefetch parameters (i.e., the number of file blocks prefetched ahead of the current access point and the number of blocks prefetched at a time). Figure 6 summarizes the results of these experiments.

With large interaccess delays, moderate prefetching suffices to service all requests from the PPFS file cache. As the interaccess time declines and the aggregate request rate increases, PPFS must prefetch a

Category	Category Features			
Read/Write	Read Only	Write Only	Read-Update-Write	Read/Write Nonupdate
Sequentiality	Sequential	1-D Strided	2-D Strided	Variably Strided
Request Sizes	Uniform		Variable	

Table 1: Qualitative Input/Output Classification



(a) UNIX (Non-adaptive)

(b) PPFS (Adaptive)

Figure 5: File Write Durations (Pathfinder on Sun SPARC 670)

larger number of blocks to maintain high cache hit ratios. Figure 6 shows that an efficient operating point depends on the request size, interaccess time, and disk performance. Thus, the slower disk system on the Paragon XP/S necessitates more aggressive prefetching than on the faster IBM SP/2. Intuitively, one can automatically determine a near optimal prefetch unit, given quantitative sensor data and a qualitative classification of the access pattern. This last point is crucial — the shape of the response time curves in Figure 6 is critically sensitive to the file access pattern.

6.3 Status and Directions

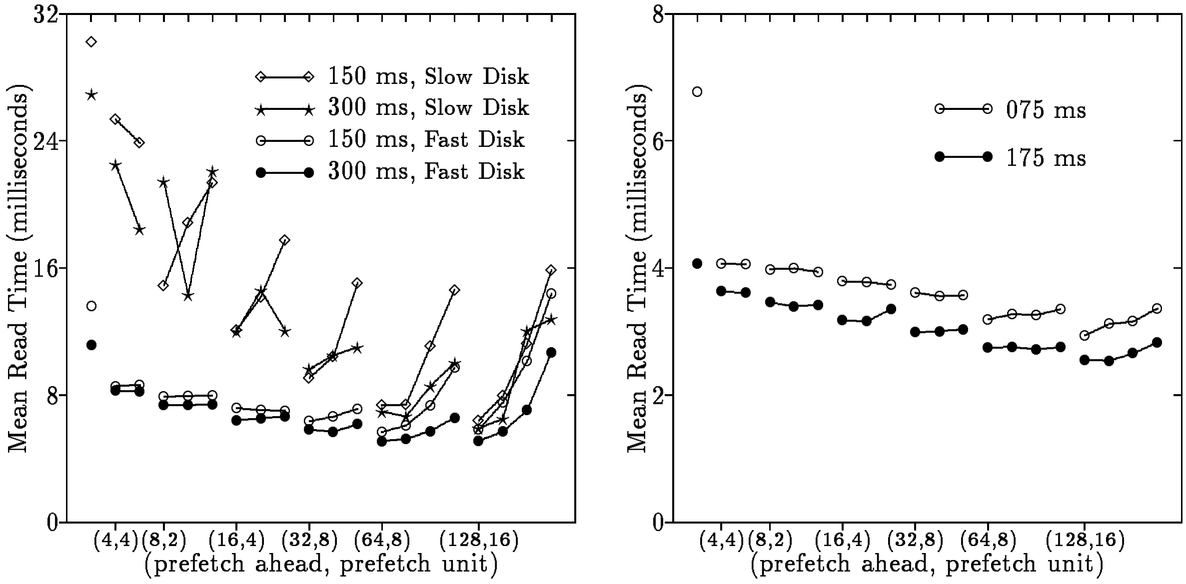
Preliminary experience with our prototype adaptive control implementation within PPFS suggests that major performance improvements are possible with performance-directed control. Based on these encouraging results, we are concurrently pursuing several research directions.

First, we are extending the adaptive PPFS prototype to more tightly couple quantitative sensor data with user assertions, automatic access pattern classification and actuators to realize a fully operational closed loop prototype. The next step is to synthesize global performance metrics from local sensor data and characterize global access patterns using local access pattern classifications. This global data will drive a set of platform-independent decision procedures for selecting file system policies.

Finally, we are beginning the detailed design of the general-purpose adaptive control infrastructure sketched in Figure 4. The control infrastructure will target three resource management domains: parallel input/output policies, thread and task scheduling, and WWW server management.

7 Interactive Adaptive Control

From a sensory perspective, the computer need not be simply a low-bandwidth medium of interaction between users and a passive software structure; instead, via immersive virtual environments users could interact directly with the executing software (e.g., resizing a file cache by "stretching" its three-dimensional representation with one's hands). In this model, the user is no longer an external observer of the system and its



(a) Intel Paragon XP/S

(b) IBM SP/2

Figure 6: PPFS Parallel Read Times (One File Server)

behavior, but replaces closed loop decision procedures described in §6 to become an active participant with the system.

7.1 Proposed Interactive Control Design

Figure 7 shows how one might assemble a virtual environment for direct direct manipulation and interactive control using the following components:

- *Virtual environment views* that can represent a hierarchy of parallel software components and their interactions, ranging from computations on geographically distributed parallel systems to module interactions within a single task.
- *Virtual environment and attribute controls* for interactive modification and adjustment of software attributes and parameters (e.g., cache sizes) that are analogous to controls on physical objects. Examples might include handles for software module movement and resizing, docking ports for software interconnection, dials and levers for specifying module parameters.
- *Direct manipulation tools* that are the virtual environment analog of a toolbox. Like physical tools, these virtual tools (e.g., a critical path finder or a dipstick for behavior sampling) augment manipulation abilities.
- *Annotation software* for marking temporal and spatial points within the virtual environment. Collectively, such annotation mechanisms would aid navigation, ease collaborative identification of important features, and provide a historical record of important events.
- *Actualization interfaces* that connect the virtual environment with dynamic performance data from executing systems and that pass information based on user manipulations to software configuration and behavior management software.

In this virtual environment, performance sensors and control actuators would permit real-time capture of performance data and modification of application or system behavior based on user manipulations within the virtual environment. Part of the environment would execute on a group of parallel or distributed systems, providing connections to sensors and actuators. The rest of the system would execute on the virtual environment platform importing and exporting these connections to and from remote systems. Annotation and performance recording software would manage user-generated multimedia data (i.e., voice, video, and text descriptions) and meta-data (e.g., extant conditions and contextual information).

The interactive control software would include a hierarchy of graphical views, ranging from procedure call graphs within a thread to geographically distributed communication among parallel systems. Via an associated suite of attribute controls and direct manipulation tools, users would be able to change software structure (e.g., by replacing software modules) and modify application parameters.

7.2 Status and Directions

We just argued that the solution to the software understanding and optimization dilemma is the development of virtual environments that allow software users and developers to directly manipulate software components while immersed in representations of software

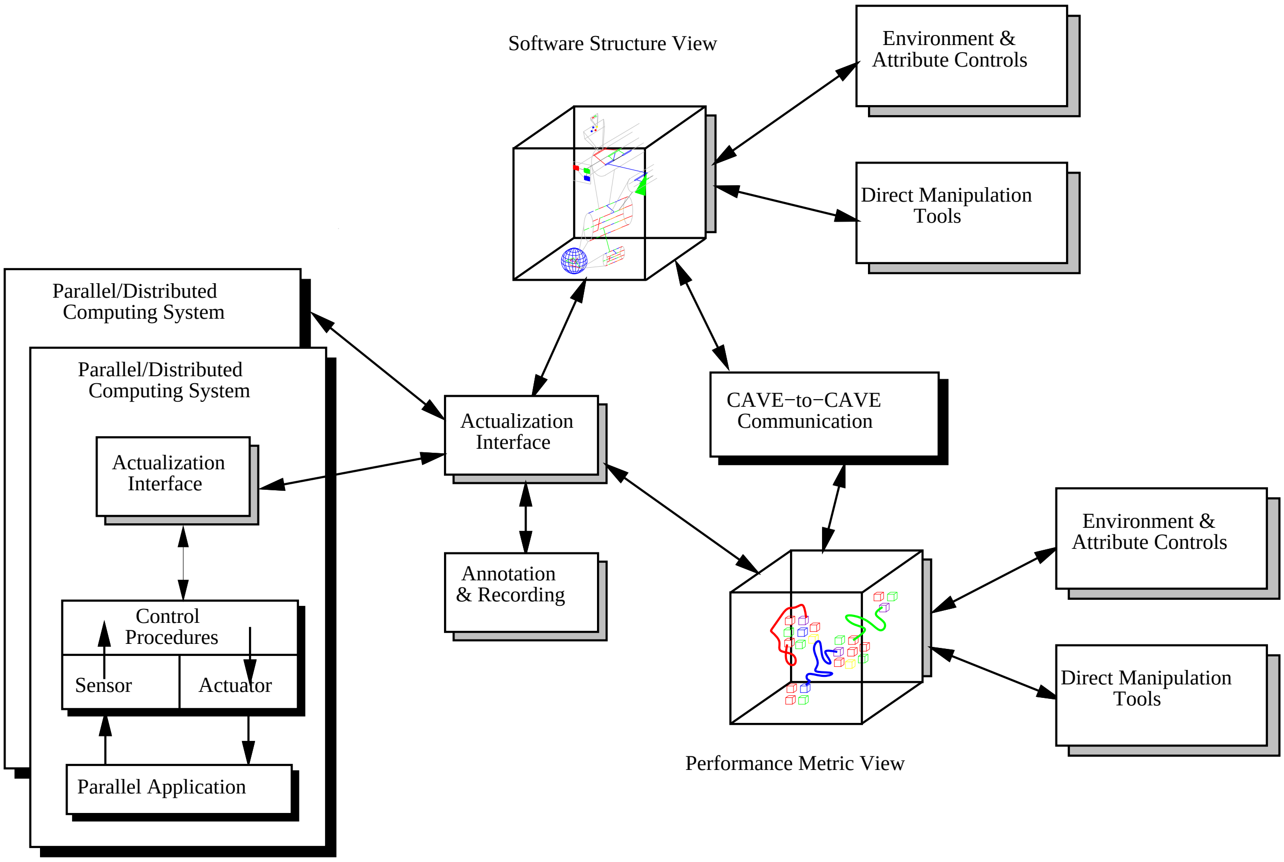


Figure 7: Interactive Resource Management Control Infrastructure

structure, behavior and real-time performance. This belief is supported by our experiences with our *Avatar* virtual environment prototype [14] and its limited support for direct manipulation of software behavior.

We have tested the *Avatar* prototype's data representations with several users and have found that virtual environment exploration can provide substantive new insights into software behavior and structure. In general, the ability to walk and fly through the data, to examine it from multiple perspectives, and to interactively change real-time display attributes has proven invaluable — we and others gained insights into behavioral and performance metric interactions that were not possible otherwise. Based on this experience, we are enhancing *Avatar* to better support direct manipulation and to tighten its coupling to the PPFS adaptive control prototype.

8 Related Work

Several systems have been built that support application behavior steering (i.e., guiding a computation toward interesting phenomena) [5]. Typically, application behavioral steering is interactive, with an application scientist studying near real-time scientific visualizations and guiding the application code by changing key application variables. The Supercomputing '95 I-WAY [5], with immersive visualizations of scientific computations, is the most recent and notable example.

In contrast to the design of systems for steering computation behavior, there have been far fewer efforts [3, 14] to interactively steer or adaptively control application performance. Notably, Schwan [3] has developed an adaptive control library for thread-based computations on shared memory parallel systems. This system, called Falcon, allows application developers to insert software sensors in their source code. Performance data from these sensors activates actuators to change program behavior based on current conditions and measured performance. The control library automatically adjusts thread locking policies based on expected synchronization delay.

9 Conclusions

Our performance analysis and characterization data show that an increasing fraction of applications in diverse problem domains (e.g., parallel input/output and WWW access) have highly variable resource demands. This data suggests that optimizing the performance of these dynamic applications will require a judicious match of resource management policies to resource request patterns.

Because the interactions between dynamic, irregular applications and system software change during application execution, we believe that a flexible infrastructure for performance directed adaptive control must replace *a posteriori* performance analysis. Such an infrastructure would allow application and system developers to create adaptive software that can change its behavior in response to real-time varying resource demands.

In this paper, we sketched the design of both interactive and closed loop adaptive control systems and described our preliminary experiences with an adaptive control prototype for parallel input/output. This prototype, based on our PPFS user-level parallel file system, selects and configures file caching and prefetching policies using both qualitative classifications of access patterns and performance sensor data on file system responses.

In the coming months, we plan to extend the adaptive PPFS prototype to more tightly couple quantitative sensor data with user assertions, automatic access pattern classification and actuators, to realize a fully operational closed loop prototype. Concurrently, we are enhancing our *Avatar* virtual environment for direct manipulation and are tightening its coupling to the PPFS adaptive control prototype. Finally, we are beginning the detailed design of the general-purpose adaptive control infrastructure sketched in Figure 4.

Acknowledgments

Ruth Aydt polished the Pablo input/output characterization software, and Phyl Crandall was one of its early users. Will Scullin, Luis Tavera, and Keith Shields designed *Avatar*, our virtual environment for performance data immersion and control. Finally, our observations on the intellectual and political problems inherent in performance tool development arose from a series of workshops co-hosted with Ann Hayes and Margaret Simmons.

References

[1] Adve, V. S., Mellor-Crummey, J., Anderson, M., Kennedy, K., Wang, J., and Reed, D. A. An Integrated Compilation and Performance Analysis Environment for Data Parallel Programs. In *Proceedings of Supercomputing '95* (Dec. 1995).

[2] Crandall, P. E., Aydt, R. A., Chien, A. A., and Reed, D. A. Characterization of a Suite of Input/Output Intensive Applications. In *Proceedings of Supercomputing '95* (Dec. 1995).

[3] Gu, W., Eisenhauer, G., Kraemer, E., Schwan, K., Stasko, J., Vetter, J., and Mallavarupu, N. Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs. In *Proceedings of the 5th Symposium of the Frontiers of Massively Parallel Computing* (Feb. 1995), pp. 422–429.

[4] Huber, J. V., Elford, C. L., Reed, D. A., Chien, A. A., and Blumenthal, D. S. PPFS: A High-Performance Portable Parallel File System. In *Proceedings of the 9th ACM International Conference on Supercomputing* (July 1995), pp. 385–394.

[5] Korab, H., and Brown, M. D. Virtual Environments and Distributed Computing at SC'95: GII Testbed and HPC Challenge Applications on the I-WAY, Dec. 1995.

[6] Kwan, T. T., McGrath, R. E., and Reed, D. A. NCSA's World Wide Web Server: Design and Performance. *IEEE Computer* (Nov. 1995), 68–74.

[7] Lamm, S. E., Scullin, W. H., and Reed, D. A. Real-time Geographic Visualization of World Wide Web Traffic. In *Proceedings of the Fifth International World Wide Web Conference* (May 1996).

[8] Malony, A. D., and Reed, D. A. Visualizing Parallel Computer System Performance. In *Instrumentation for Future Parallel Computing System s*, M. Simmons, R. Koskela, and I. Bucher, Eds. Addison-Wesley Publishing Company, 1989, pp. 59–90.

[9] Messina, P., and Sterling, T., Eds. *Pasadena Workshop on System Software and Tools for High-Performance Computing Environments.* SIAM, Jan. 1992.

[10] Poole, J. T. Scalable I/O Initiative. California Institute of Technology, Available at `http://www.ccsf.caltech.edu/SIO/`, 1996.

[11] Reed, D. A. Performance Instrumentation Techniques for Parallel Systems. In *Models and Techniques for Performance Evaluation of Computer and Communications Systems*, L. Donatiello and R. Nelson, Eds. Springer-Verlag Lecture Notes in Computer Science, 1993, pp. 463–490.

[12] Reed, D. A., Elford, C. L., Madhyastha, T., Scullin, W. H., Aydt, R. A., and Smirni, E. I/O, Performance Analysis, and Performance Data Immersion. In *Proceedings of MASCOTS '96* (Feb. 1996), pp. 1–12.

[13] Reed, D. A., and Rudolph, D. C. Experiences with Hypercube Operating System Instrumentation. *International Journal of High-Speed Computing* (Dec. 1989), 517–542.

[14] Reed, D. A., Shields, K. A., Tavera, L. F., Scullin, W. H., and Elford, C. L. Virtual Reality and Parallel Systems Performance Analysis. *IEEE Computer* (Nov. 1995), 57–67.

[15] Simmons, M. L., Hayes, A. H., Brown, J. J., and Reed, D. A., Eds. *Debugging and Performance Tuning for Parallel Computing Systems.* IEEE Computer Society Press, 1996.

[16] Smirni, E., Aydt, R. A., Chien, A. A., and Reed, D. A. I/O Requirements of Scientific Aapplications: An Evolutionary View. In *submitted for publication* (1996).