A Framework for Adaptive Storage
Input/Output on Computational Grids*

Huseyin Simitci, Daniel A. Reed, Ryan Fox, Mario Medina, James Oly,
Nancy Tran, and Guoyi Wang

Department of Computer Science
University of Illinois
Urbana, Illinois 61801, USA
{simitci,reed,r-fox,c-medina,jamesoly,n-tran1,gwang2}@cs.uiuc.edu
WWW Home Page http://www-pablo.cs.uiuc.edu

Abstract. Emerging computational grids consist of distributed collections of heterogeneous sequential and parallel systems and irregular applications with complex, data dependent execution behavior and time varying resource demands. To provide adaptive input/output resource management for these systems, we are developing PPFS II, a portable parallel file system. PPFS II supports rule-based, closed loop and interactive control of input/output subsystems on both parallel and wide area distributed systems.

1   Introduction

Current users of homogeneous parallel systems often complain that it is difficult to achieve a high fraction of the theoretical performance peak. Distributed collections of heterogeneous systems (i.e., the computational grid), introduce new and more complex application tuning and performance optimization problems. Moreover, the sensitivity of performance to slight changes in application code, together with the large number of potential application performance problems (e.g., load balance, data locality, or input/output) and continually evolving system software, make application tuning complex and often counterintuitive.

Of the range of grid performance problems, the lack of high-performance input/output is one of the most debilitating. Not only is the performance gap between processors and secondary storage devices continuing to increase, the input/output performance of parallel systems is exceedingly sensitive to slight changes in application behavior. The time-varying availability of distributed resources and changing application demands of the computational grid exacerbates these problems. Hence, it is extraordinarily difficult to choose a single set of file

system policies that yield high performance for all applications and resource demands.

Instead, by integrating dynamic performance instrumentation and malleable file system policies with adaptive control systems, a flexible runtime system could automatically configure resource management policies and parameters based on application request patterns and observed system performance. Such a dynamic system will allow applications and runtime libraries to adapt to changing hardware and software characteristics.

Recent experiences with flexible input/output policies [11] and adaptive runtime systems [18, 7] has shown that adapting to changing resource demands and availability can yield order of magnitude performance improvements. To provide a flexible testbed for high-performance input/output experiments, we have designed and implementation a prototype, second generation portable parallel file system, PPFS II, that includes real-time, adaptive policy control. PPFS II is designed to work atop either parallel systems or PC and workstation clusters.

The remainder of this paper is organized as follows. In §2, we outline the general design of PPFS II, followed in §3–§4 by a description of adaptive file caching and redundant disk striping. In §5, we describe *Autodriver*, a toolkit for interactive visualization of PPFS II behavior and steering of PPFS II policies. This is followed in §6 by a discussion of new input/output arrival traffic pattern forecasting techniques using ARIMA time series analysis and Hidden Markov models. Finally, §7 summarizes the current state of PPFS II and outlines directions for additional research.

2   PPFS II File System Architecture

Figure 1 illustrates the high-level components of our PPFS II prototype. Globus [7] provides communication and system interaction within a framework for dynamic systems on distributed computational grids. Atop Globus, we have implemented a toolkit for real-time adaptive control of distributed computations. This *Autopilot* toolkit provides a flexible set of performance sensors, decision procedures, and policy actuators to realize adaptive control of applications and resource management policies.

In PPFS II, performance sensors capture real-time performance data from distributed input/output components, decision procedures choose and configure input/output policies, and policy actuators implement policy decisions. All PPFS II sensors and actuators are registered with the *Autopilot* manager, which serves as a central place to publish and query available PPFS II components.

The PPFS II metadata manager organizes the distributed components of PPFS II files, noting data locations and access privileges. In turn, storage servers provide a uniform interface to native file systems available on the systems where they are executing. Decision server contains the fuzzy logic rule bases and control mechanisms for closed-loop adaptive control of the PPFS II file system. Finally, the Java-based *Autodriver* interface provides interactive monitoring and control.
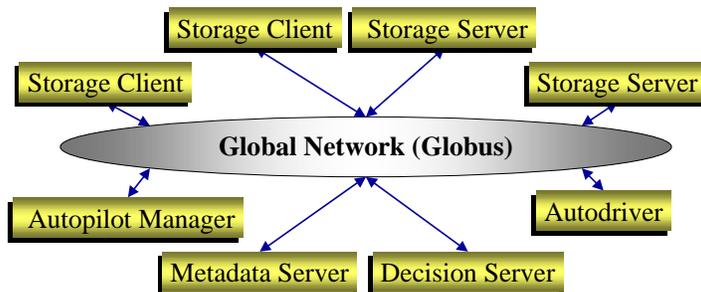
Figure 1. PPFS II Distributed Grid Architecture.

Using the *Autopilot* toolkit, PPFS II integrates real-time access pattern classification software [12] and a fuzzy logic decision mechanism [18] for dynamically identifying access patterns and adaptively choosing file system policies. The fuzzy logic infrastructure accepts access pattern classifications and performance measures and uses a configurable rule base to choose file policies. Currently, rule bases for adaptive caching and striping are implemented.

Application clients access the file system via the Scalable I/O (SIO) Initiative's low-level API [4]. This API is intended as a portable interface for implementing higher level I/O libraries (e.g., MPI-IO [23]). Hence, the SIO API includes routines that can express complex parallel input/output patterns easily (e.g., strided accesses). It also includes a "hint" interface applications can use to specify both quantitative (i.e., offsets and sizes) and qualitative (.i.e., sequential or random) access patterns. For portability, PPFS II also supports standard UNIX file system calls through converters that are implemented atop the SIO API.

3 Adaptive File Caching

As discussed earlier, PPFS II relies on the *Autopilot* adaptive control infrastructure to implement an adaptive file caching scheme. Figure 2 shows the components of this adaptive caching system. Below, we describe this structure in more detail and discuss the performance benefits of adaptive caching.

3.1 PPFS II/Autopilot Interactions

PPFS II exploits *Autopilot*'s suite of configurable, lightweight sensors to capture quantitative and qualitative performance data for input/output policy decisions. To balance the communication cost of transmitting large volumes of raw performance data against local computation overhead, *Autopilot* allows one to attach data transformation functions to each sensor. These attached functions can compute simple statistics from raw data (e.g., sliding window averages) or
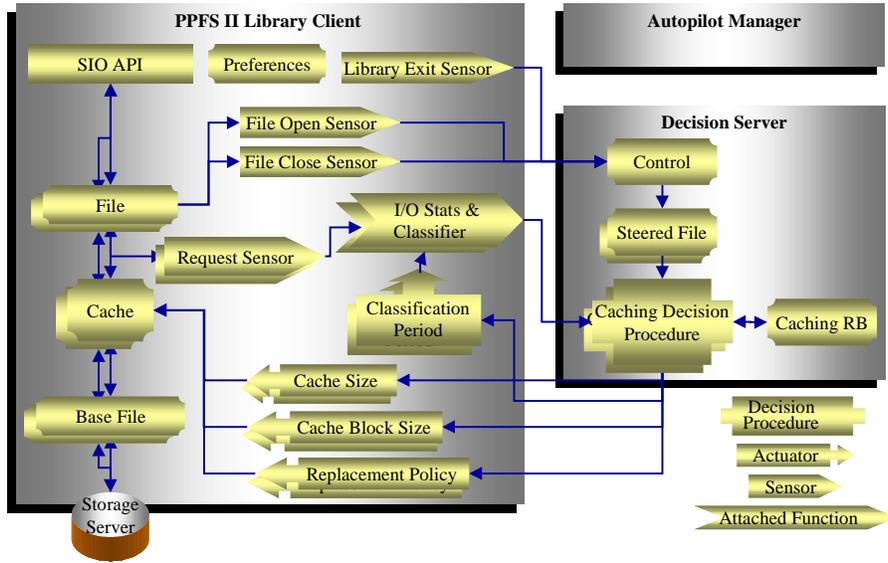
Figure 2. PPFS II Adaptive Caching System Components.

more complex transformations (e.g., computing qualitative access input/output patterns from request streams).

In PPFS II, two *Autopilot* sensors inform the decision server whenever a file is opened or closed by a remote client. Via this sensor notification, the decision server can choose caching and striping policies for the file's active lifetime.

In addition, each input/output request triggers a request sensor whose attached function calculates input/output statistics (e.g., request rates and sizes) and classifies the file access pattern in real-time. This classification exploits a trained neural network to classify access patterns as uniform or variable size, sequential or strided, and read-only, write-only, or read-update-write. Using the output of this sensor, the decision server can change caching and striping policies based on access pattern attributes and request rates.

The decision server controls the file system via four *Autopilot* actuators. These actuators accept remote inputs and can dynamically modify file system variables (e.g., cache size or cache block size) and to change cache management policies. Currently, the file system cache implements Least-Recently-Used (LRU), Most-Recently-Used (MRU) and random replacement policies, all of which can be changed during PPFS II execution. In summary, PPFS II implements a closed loop adaptive control system that dynamically tailors the file system cache's parameters to application's request patterns and observed input/output performance.

3.2 Adaptive Caching Experiments

To assess the performance benefits of adaptive input/output policy control, we have tested our PPFS II prototype in a variety of hardware and software configurations. These experiments verified the effectiveness of real-time performance monitoring and access pattern detection for policy selection and control with both single and multiple process benchmarks.

Below, we describe the results of one set of experiments conducted on a cluster of Linux PCs, each with three Ultra SCSI disks. To simplify explanation, we illustrate PPFS II's closed loop behavior using the interactive *Autodriver* interface, described in §5, to visualize and manipulate PPFS II behavior. In this example, shown in Figure 3, a benchmark program writes 128 KB blocks to a file striped over eight storage servers.

In the figure, the write response time sensor shows the application response times in microseconds. Initially, client caching is disabled by default. After activating caching via the cache mode actuator, file writes are buffered in the cache until it fills.

Using the cache block size actuator shown at the bottom of Figure 3, we increase the cache block size 512 KB. Subsequently, most of the application writes are buffered in the cache and written to secondary storage in groups large enough to benefit from striping across multiple disks. This significantly reduces average response time.

4 Adaptive, Redundant Disk Striping

Because disk capacities continue to rise faster than transfer rates, only input/output parallelism [19] can provide the bandwidth needed by high-performance parallel and and distributed applications. This technique, known as disk striping, is the fundamental idea behind striped file systems [8, 6] and disk arrays [15]. However, matching the distribution of data across storage devices to system utilization has proven critical to obtaining high performance.

Previously, Chen and Patterson [3] and Scheuerman *et al* [20] concluded that optimal striping units (i.e., the amount of continuous data written on each disk) were strongly dependent on the degree of resource contention. Intuitively, at low loads, aggressive striping trades throughput for reduced response times. At higher loads, less aggressive striping sacrifices response time for higher throughput.

Building on the results of our earlier parallel input/output analysis [12, 5], our ongoing PPFS II experiments have continued to explore the match of file striping to access policies and system loads. To understand the effects of file striping parameters on PPFS II performance, we have used a suite of SIO [4] benchmarks. As an example, Figure 4 shows the average response times of random, 128 KB requests, when eight clients read a 1 GB file.

The figure illustrates the complex interplay of inter-request delay, number of disks used for striping, and request size. There is more than two-fold performance differential based on stripe width. When one includes competing access
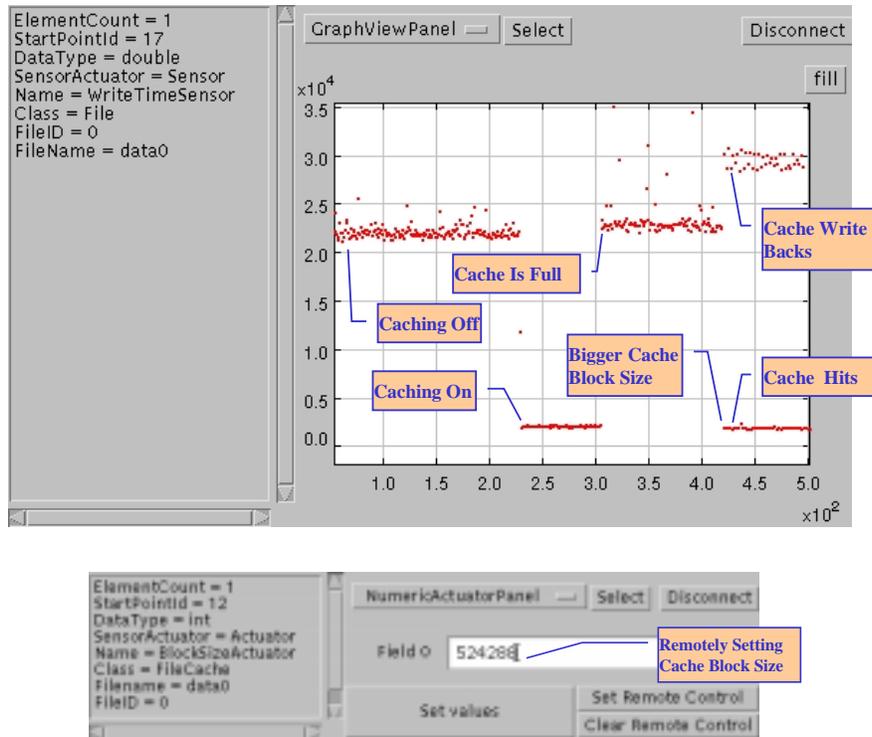
Figure 3. Annotated Write Response Time Control.

to multiple files, disks connected by variable latency, wide area networks, and multiple access patterns and request sizes, the complexity of the performance optimization problem becomes clear.

With these and other experiences, we have developed analytic queueing models of disk striping. This model considers request patterns, disk attributes, and network characteristics to predict the response times for striped requests. For detailed derivation and analysis of this model, see [21]. Based on this model, we have developed a fuzzy logic rule base for adaptive disk striping. This rule base distributes requests to multiple disks during periods of low system load to reduce the individual request response times. At high load, the rule base reduces the striping width to maximize system throughput.

These adaptive striping optimizations generate multiple redundant copies of the parallel files — each optimized for certain access and system characteristics. Simply put, redundant striping trades access disk capacity for higher input/output throughput.
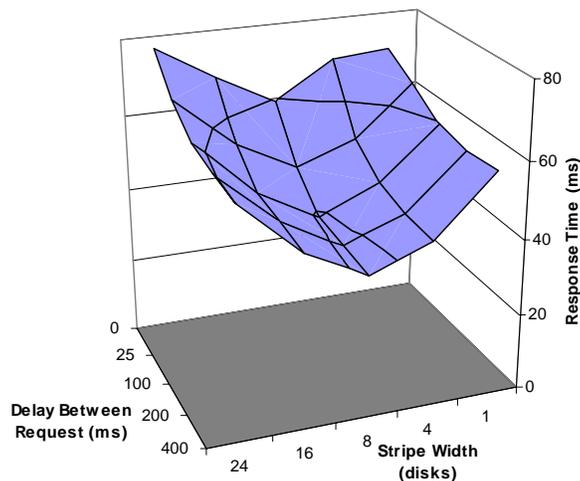
Figure 4. PPFS II Cluster I/O Elapsed Time (1 GB File Read)

## 5 Interactive PPFS II Control with Autodriver

Although PPFS II is primarily intended to operate as an adaptive, closed loop software system, it can also be steered interactively. These interactive controls allow file system designers to understand the behavior of PPFS II policies and to augment PPFS II software controls with user steering. *Autodriver* is our portable interface for desktop and mobile interaction with *Autopilot* and PPFS II sensors and actuators.

Written largely in Java to enhance portability, *Autodriver* queries *Autopilot* managers to obtain a list of available sensors and actuators, and allows users to select the system aspects they wish to visualize and control. *Autodriver* includes both textual and graphical views of connected sensors. Moreover, actuator values can be changed through either a generic value entry panel or via views customized for specific actuator types.

As an example, Figure 5 shows the *Autodriver* user interface. The window in the figure displays the sensors and actuators currently registered with the *Autopilot* manager. As sensors and actuators are created and destroyed, this list is automatically updated.

Selecting a sensor or actuator displays detailed information for that item in the lower part of the window. Double-clicking on a sensor or actuator creates a new window that shows, respectively, sensor data values or an input panel actuator control.

As a second example, the upper window in Figure 3 shows data for a PPFS II sensor named *WriteTimeSensor*. This view is a simple scrolling scatterplot of sensor values. One can zoom closer to focus on areas of particular interest or save
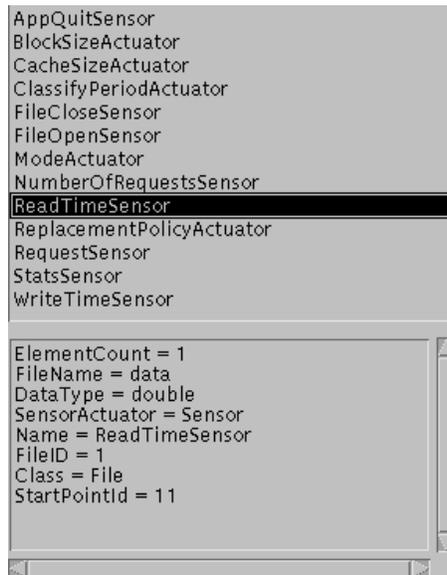
Figure 5. Autodriver Control Interface.

the data in a file for subsequent analysis. Finally, the lower window in Figure 3 illustrates remote actuator configuration and control.

Although Figure 3 illustrates use of *Autodriver* with our PPFS II file system prototype, *Autodriver* is domain independent. One can connect to any sensors and actuators registered with the *Autopilot* on either a local or remote system. By continuing to extend *Autodriver*'s Java interface, our goal is to make PPFS II and *Autopilot* sensors and actuators accessible anywhere and any time (e.g., via handheld devices and wireless networks).

6 New Access Pattern Classification Techniques

Currently, PPFS II uses artificial neural networks (ANNs) to automatically classify file access patterns during application execution. Building on the success of ANNs, we are exploring two complementary, more powerful approaches to access pattern classification based on time series analysis and hidden Markov models (HMMs). Each is described below.

6.1 Time Series Arrival Pattern Forecasting

To create ANNs, access pattern samples are used to train feedforward artificial neural networks [12]. Once trained, PPFS II read/write sensors invoke the ANN periodically for *qualitative* access pattern classification (e.g., sequential or strided).

(a) Unfiltered (b) Filtered
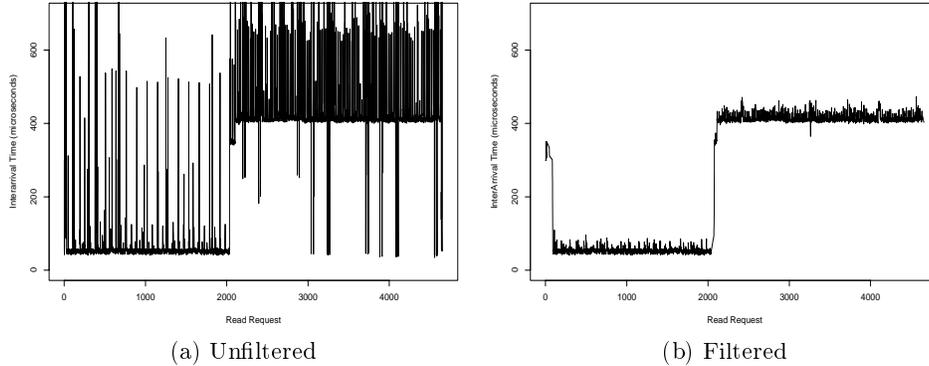
**Figure 6.** PRISM Read Request Interarrival Times (Processor 0).

As a complementary approach to ANNs, Box-Jenkins ARIMA (Autoregressive Integrated Moving Average) time series analysis [2] can be used to create *quantitative* forecasts of future access patterns and trends. In particular, we are investigating ARIMA's ability to predict the *burstiness* of input/output requests. Such Quantitative forecasts of input/output arrival patterns would allow PPFS II to plan prefetching based on anticipated, rather than observed need — prefetching too late misses the window of opportunity for overlapping computations with disk input/output [16].

ARIMA is a stochastic data analysis and modeling methodology that can extract a wide variety of data patterns and seasonal trends, forecasting new behavior from past observations [1, 24]. As a test of ARIMA's applicability to input/output pattern forecasting, we have applied ARIMA to input/output trace data from the Intel Paragon XP/S at Caltech, captured using our Pablo instrumentation toolkit [17].

The Paragon XP/S PRISM application simulates the dynamics of turbulent flow using the three-dimensional Navier-Stokes equations [9, 22]. To study input/output request burstiness, we extracted the interarrival times of file read requests from the PRISM traces for processor zero. The request interarrival times for other processors exhibit similar behavior. Figure 6(a) shows the sequenced of interarrival times.

Inspection reveals that the data is very noisy, with many outliers that obscure the underlying data pattern.[1] After filtering with an approximate conditional mean robust filter, we obtain the data in Figure 6(b), which clearly shows two phase transitions, one near the beginning of execution and one a second near observation 2030. For simplicity's sake, we concentrate on the interval between the first and second transitions.

---

[1] We documented the wide variability of input/output times on the Intel Paragon XP/S in earlier studies [5].
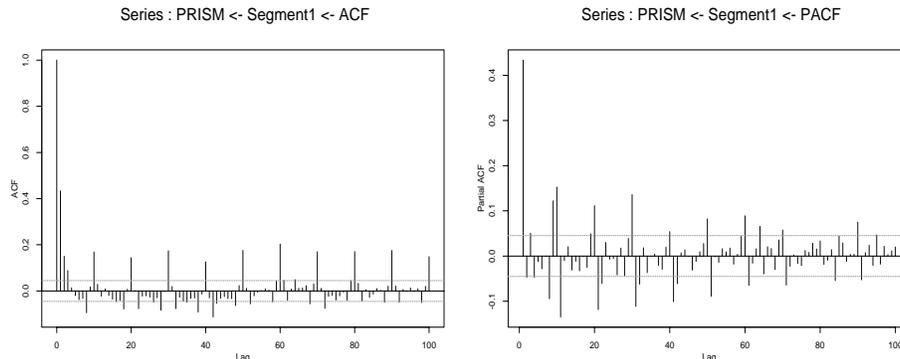
Figure 7. ACF and PACF of Filtered PRISM Series.

Fitting an ARIMA model to a time series consists of three basic steps: (a) model identification, (b) parameter estimation and diagnostic checking, and (c) forecasting. We briefly describe application of these steps to the filtered PRISM data. Space precludes a complete description of the ARIMA process; for additional details, see [24].

Model Identification and Parameter Estimation The two fundamental tools used by ARIMA to expose and identify data patterns are the autocorrelation function (ACF) and the partial autocorrelation function (PACF). ACF measures the linear relationships between pairs of observations, whereas PACF measures the conditional correlation by removing the effects of intervening observations.

Figure 7 shows the estimated ACF and PACF for the filtered PRISM series. The dotted band about zero represents a 95 percent confidence interval. In the ACF plot, significant spikes extend above the dotted band, appearing at regular intervals of ten requests. This correlation pattern indicates that the series is seasonal/periodic with a period of size 10.

The PRISM data also has a non-seasonal pattern, shown by the single large spike at lag one in the PACF plot, followed by a cutoff to insignificant values below the dotted band. Combining the seasonal and nonseasonal patterns gives an $ARIMA(1, 0, 0)(0, 1, 1)^{10}$ model. Additional parameter estimation yields $\phi_1 = 0.4964$ for the nonseasonal part and $\Theta_1 = 0.9930$ for the seasonal part.

Intuitively, these seasonal and non-seasonal patterns in the PRISM data are caused by loops and data-dependent control flow in the PRISM code. Seasonal data accrues from nested loops, whereas data-dependent control flow creates aperiodic behavior. The ARIMA model successfully captures both behaviors.

Model Forecasting Given an ARIMA model, it can be used to forecast request patterns by supplying the model with real-time performance data. To validate the forecast, we supplied the estimated ARIMA model with a shortened PRISM series with the final sixty input/output requests omitted.
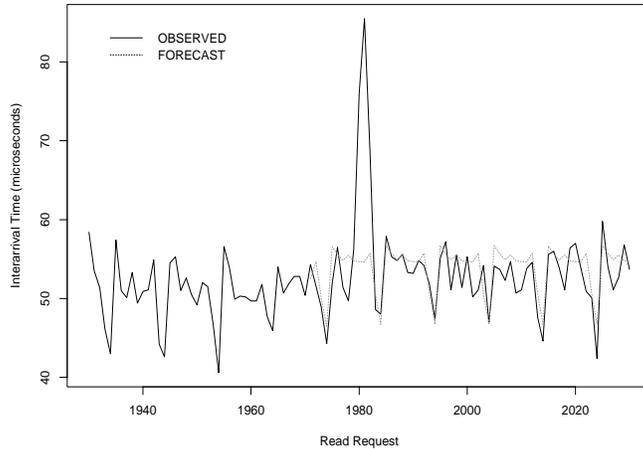
Figure 8. ARIMA Forecast Validation.

Figure 8 compares the forecasts for these sixty observations with the observed PRISM values. With one exception, the predictions are quite accurate, tracking both the seasonal and non-seasonal patterns.

Based on this encouraging match, we are working to develop on-line techniques for constructing ARIMA and other time series models. Via these on-line models, the PPFS II decision procedures can configure cache prefetching schemes that anticipate input/output request bursts, retrieving the desired disk blocks before the requests arrive.

6.2 Hidden Markov Model Access Pattern Forecasting

Recently, several groups have explored optimal caching and prefetching policies when parallel file access patterns are fully known [10]. However, such foreknowledge is often impossible to obtain. For instance, an application's access pattern may change across executions due to data dependencies or because differing system loads may change scheduling decisions for input/output tasks. Even if the access stream is known precisely, varying network latencies and loose client synchronization can change the ordering of requests at input/output servers. In these and similar situations, probabilistic access models can prove useful.

To explore the utility of probabilistic models, we have developed hidden Markov Model (HMM) access pattern classification techniques [14, 13]. In standard Markov models, the next labeled output state depends solely on the current state. Thus, a typical Markov model for file systems uses a state for each disk block, with state labels corresponding to block identifiers and transitions representing the probability that the source and destination states (blocks) are

accessed in sequence. HMMs generalize Markov models by associating a probability with each state — a new output state is chosen using the probability distribution when the state is visited.

One can construct Markov models or HMMs from a trace of application input/output requests, either offline using an actual trace or incrementally during application execution. Once a Markov model is built for an application's request pattern, it can be used to control caching and prefetching decisions. For example, by computing high probability paths through the transition graph, prefetching algorithms can retrieve non-sequential groups of blocks that are most likely to be accessed in the future. With ancillary data on block interreference times, one can identify and remove from the file cache those blocks that will be referenced the longest in the future

Building on our preliminary experiences with HMM classification, we are adding HMMs to PPFS II. By enabling PPFS II to manage complex access patterns, these algorithms will allow PPFS II to adapt to changing conditions.

7 Conclusions and Future Work

PPFS II combines advanced techniques for adaptive control of distributed, parallel file systems. Our experiments with PC and workstation clusters have shown that automatic, rule-based adaptive control can yield significant performance improvements over statically configured input/output policies. However, these studies also revealed that optimal disk striping parameters (i.e., the striping unit and width) are strongly dependent on the interplay of application request rates and hardware configurations. Consequently, striping configurations, like caching and prefetching policies must be chosen with care if one is to maximize possible performance benefits.

To identify application input/output patterns, we continue to explore both time series and hidden Markov model (HMM) techniques. Our preliminary studies suggest that ARIMA time series analysis can accurately forecast an application's input/output patterns. However, input/output request models are currently constructed offline. We are investigating dynamic model identification methods that can adaptively derive new models using real-time performance data. Similarly, we believe HMMs can successfully build probabilistic models access pattern using previous executions. This generality will allow automatic classification of arbitrary access patterns.

Finally, the closed loop and interactive performance steering techniques developed for PPFS II are applicable to a broader range of resource domains than just input/output. We believe they can be profitably applied to task scheduling and network protocol configuration. Exploring application of adaptive techniques to these domains is the subject of ongoing work.

Acknowledgments

References

1. ABRAHAM, B., AND LEDOLTER, J. *Statistical Method for Forecasting*. John Wiley and Sons, 1983.

2. BOX, G. E., AND JENKINS, G. M. *Time Series Analysis Forecasting and Control*. 2nd ed. San Francisco: Holden-day, 1976.

3. CHEN, P. M., AND PATTERSON, D. A. Maximizing Performance in a Striped Disk Array. In *Proceedings of the 17th Annual International Symposium on Computer Architecture* (1990), pp. 322–331.

4. CORBETT, P. F., PROST, J.-P., DEMETRIOU, C., GIBSON, G., RIEDEL, E., ZELENKA, J., CHEN, Y., FELTEN, E., LI, K., HARTMAN, J., PETERSON, L., BERSHAD, B., WOLMAN, A., AND AYDT, R. Proposal for a Common Parallel File System Programming Interface Version 1.0. http://www.cs.arizona.edu/sio/api1.0.ps, Nov. 1996.

5. 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).

6. DIBBLE, P. C., SCOTT, M. L., AND ELLIS, C. S. Bridge: A High-Performance File System for Parallel Processors. In *Proc. 8th Int'l. Conf. on Distr. Computing Sys.* (San Jose, CA, jun 1988), pp. 154–161.

7. FOSTER, I., AND KESSELMAN, C. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications and High Performance Computing 11*, 2 (Summer 1997), 115–128.

8. HARTMAN, J. H., AND OUSTERHOUT, J. K. The Zebra Striped Network File System. *ACM Transactions on Computer Systems 13*, 3 (Aug. 1995), 274–310.

9. HENDERSON, R. D., AND KARNIADAKIS, G. E. Unstructured Spectral Element Methods for Simulation of Turbulent Flows. *Journal of Computational Physics 122(2)* (1995), 191–217.

10. KIMBREL, T., TOMKINS, A., PATTERSON, R. H., BERSHAD, B., CAO, P., FELTEN, E., GIBSON, G., AND KARLIN, A. R. A Trace-Driven Comparison of Algorithms for Parallel Prefetching and Caching. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation* (1996), pp. 19–34.

11. MADHYASTHA, T. M., ELFORD, C. L., AND REED, D. A. Optimizing Input/Output Using Adaptive File System Policies. In *Proceedings of the Fifth Goddard Conference on Mass Storage Systems and Technologies* (Sept. 1996), pp. II:493–514.

12. MADHYASTHA, T. M., AND REED, D. A. Intelligent, Adaptive File System Policy Selection. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation* (Oct 1996), IEEE Computer Society Press, pp. 172–179.

13. MADHYASTHA, T. M., AND REED, D. A. Exploiting Global Input/Output Access Pattern Classification. In *Proceedings of SC '97: High Performance Computing and Networking* (San Jose, Nov. 1997), IEEE Computer Society Press.

14. MADHYASTHA, T. M., AND REED, D. A. Input/Output Access Pattern Classification Using Hidden Markov Models. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems* (San Jose, CA, Nov. 1997), ACM Press, pp. 57–67.

15. PATTERSON, D., CHEN, P., GIBSON, G., AND KATZ, R. H. Introduction to Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of IEEE Compcon* (Spring 1989), pp. 112–117.

16. PATTERSON, R., GIBSON, G., GINTING, E., STODOLSKY, D., AND ZELENKA, J. Informed Prefetching and Caching. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, CO.* (December 1995), pp. 79–95.

17. REED, D., AYDT, R., NOE, R., ROTH, P. C., SHIELDS, K. A., SCHWARTZ, B., AND TAVERA, L. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proceedings of the Scalable Parallel Libraries Conference. IEEE Computer Society* (1993), pp. 104–113.

18. RIBLER, R. L., VETTER, J. S., SIMITCI, H., AND REED, D. A. Autopilot: Adaptive Control of Distributed Applications. In *Proceedings of HPDC 7* (July 1998).

19. SALEM, K., AND GARCIA-MOLINA, H. Disk Striping. In *Proceedings of the 2nd International Conference on Data Engineering* (Feb. 1986), ACM, pp. 336–342.

20. SCHEUERMANN, P., WEIKUM, G., AND ZABBACK, P. Data Partitioning and Load Balancing in Parallel Disk Systems. *The VLDB Journal 7*, 1 (Feb. 1998), 48–66.

21. SIMITCI, H., AND REED, D. A. Adaptive Disk Striping for Parallel Input/Output. In submitted for publication (1999).

22. SMIRNI, E., AND REED, D. Workload Characterization of Input Output Intensive Parallel Applications. In *Proceedings of the Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Springer-Verlag Lecture Notes in Computer Science* (June 1997), vol. 1245, pp. 169–180.

23. THE MPI-IO COMMITTEE. MPI-IO: A Parallel File I/O Interface for MPI, April 1996. Version 0.5.

24. WEI, W. S. *Time Series Analysis Univariate and Multivariate Methods.* Addison-Wesley, 1990.