Real-time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids[1]

Jeffrey S. Vetter
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551 USA
vetter3@llnl.gov

Daniel A. Reed
Department of Computer Science
University of Illinois
Urbana, Illinois 61801 USA
reed@cs.uiuc.edu

1. Introduction

The scope of high-performance computing is rapidly expanding from single parallel systems to clusters of heterogeneous sequential and parallel systems. Moreover, as applications become more complex, they grow more irregular, with data-dependent execution behavior, and more dynamic, with time-varying resource demands. Consequently, even small changes in application structure can lead to large changes in observed performance. This performance sensitivity is a direct consequence of the complexity of resource interaction and a clear indication that resource management must evolve with applications. With the emergence of computational grids and their ever-changing resource base, resource management must become more flexible and responsive to changing resource availability and resource demands.

To support creation of nimble applications for computational grids, we believe one must not only tightly integrate compilers, languages, libraries, algorithms, problem-solving environments, runtime systems, schedulers, and tools but also eliminate the barrier that separates program creation from execution and post-mortem optimization. This view is based on our experience building performance instrumentation and analysis tools for parallel systems [5, 6, 8], integrating runtime measurement and deep compile-time analysis [1], and from an ongoing series of discussions application, compiler, library, and runtime system developers.[2]

In this integrated model, high-level problem solving environments (PSEs) allow users to compose programs from configurable modules, each with a performance specification interface that describes the expected module behavior as a function of available resources. Using PSE specifications, whole program compilers generate configurable object programs that include embedded performance instrumentation and support for runtime validation of performance specifications. During execution, these configurable programs dynamically adapt to changing resource availability, invoking compilers to generate code variants optimized for current conditions and exploiting adaptive runtime systems to negotiate program behavior and resource management.

To support dynamic performance adaptation and distributed optimization in the grid environment, we are building a suite of performance instrumentation, analysis and presentation tools that includes distributed performance sensors and resource policy actuators, fuzzy logic rule bases for adaptive control, and immersive visualization systems for real-time visualization and direct manipulation of software behavior. With this context, the remainder of this paper is organized as follows. In §2, we

[1] This work was supported in part by the Defense Advanced Research Projects Agency under DARPA contracts DABT63-94-C0049 (SIO Initiative), F30602-96-C-0161, and DABT63-96-C-0027 by the National Science Foundation under grants NSF CDA 94-01124, ASC 97-20202, and EIA 99-77284, by the NSF PACI Computational Science Alliance Cooperative Agreement, and by the Department of Energy under contracts DOE B-341492, W-7405-ENG-48, and 1-B-333164.
[2] In particular, the GrADS team of Fran Berman, Jack Dongarra, Ian Foster, Dennis Gannon, Lennart Johnsson, Ken Kennedy, Carl Kesselman, and Rich Wolski has shared stimulating ideas on all aspects of computational grids.

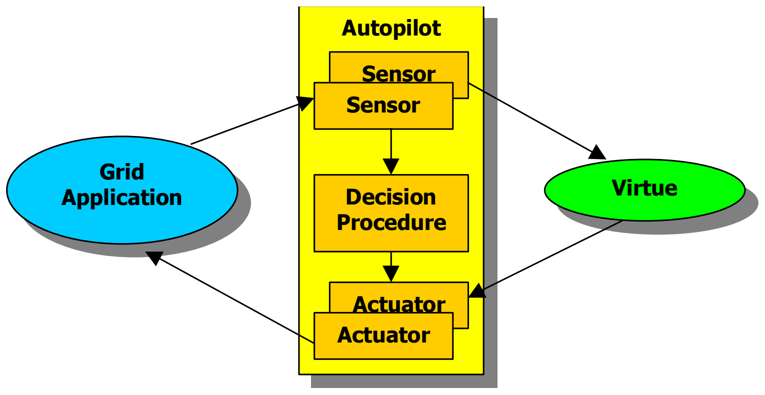

Figure 1 Distributed Software Visualization and Control

summarize our approach to building grid performance tools. This is followed in §3–§4 by a description of our grid instrumentation and visualization toolkits, respectively called Autopilot and Virtue. In §5, we describe related work, followed in §6 by a summary of future directions.

2. Grid Performance Tools

To create a performance measurement and analysis substrate for computational grids, we are developing two interoperable toolkits for measurement and visualization. Autopilot, our distributed performance measurement and resource control system, is based on our experiences using the Pablo performance toolkit for application measurement [5,6], I/O characterization [10], and WWW traffic analysis.

Autopilot is complemented by Virtue, an immersive environment that accepts real-time performance data from Autopilot and allows users to change software behavior and resource policies by directly manipulating representations of software structure and dynamics. In tandem, Autopilot and Virtue allow application developers and performance analysts to capture, analyze, visualize and steer distributed computations that execute on thousands of distributed processors.

Autopilot and Virtue are built atop portions of the Pablo toolkit [5,6] and the Globus runtime system. Both Autopilot and Virtue use the Globus toolkit [1] for wide area communication and task management. Because Globus supports multiple communication protocols, including shared memory, MPI, and TCP/IP, it unites parallel systems, PC and workstation clusters, and wide-area networks, allowing Autopilot and Virtue to interoperate via in a single communication model. In addition, Autopilot and Virtue both use the Pablo Self-Defining Data Format (SDDF) as their medium of information exchange – Autopilot generates runtime information in SDDF format and Virtue accepts SDDF data for immersive display.

Because Virtue displays are coupled to Autopilot sensors and actuators using SDDF, Virtue can display real-time performance data from the geographically distributed components of a grid application and allow users to change software behavior simply by grasping and manipulating representations of computation. As shown in Figure 1, these manipulations are translated automatically to Autopilot actuator commands for the relevant software component. In this model, the user replaces Autopilot's decision procedures as an intelligent steering agent.

3. Autopilot: Distributed Performance Optimization

On heterogeneous collections of geographically distributed computing resources, the execution context may not be repeatable across program executions and resource availability may change during execution. In such chaotic environments, only real-time, distributed measurement and dynamic optimization can adapt to changing application resource demands.

To meet these demands, Autopilot [9] integrates application and system instrumentation with resource policies and distributed decision procedures. The resulting closed loop adaptive control system can automatically configure resources based on application request patterns and system performance. In consequence, application and runtime library developers can create nimble software that optimizes performance in response both changing demands and resource availability.

3.1 Toolkit Structure

The Autopilot library contains the following components

- Distributed performance sensors that capture quantitative application and system performance data, with configurable options for data buffering, local reduction, and transmission.
- Software actuators that can enable and configure application behavior and resource management policies.
- Qualitative behavioral classification tools, based on hidden Markov models and artificial neural networks, which describe application resource request patterns (e.g., large read requests or short computation intervals).
- An extensible self-defining data format (SDDF) that separates the structure of sensor data from its semantics.
- Fuzzy logic decision procedures that accept performance sensor data and both choose and configure resource management policies via distributed actuators.
- Distributed name servers that support registration by remote sensors and actuators and property-based requests for sensors and actuators by remote clients.
- Sensor and actuator clients that interact with remote sensors and actuators, monitoring sensor data and issuing commands to actuators.

In this design, shown in Figure 2, performance instrumentation sensors capture and compute quantitative application and system performance metrics. Qualitative application behavior is obtained via automatic behavioral classification techniques and shared via the same sensor distribution mechanisms. The qualitative and quantitative data are used by a hierarchy of remote clients that contain decision procedures for choosing and configuring application and system resource management policies via software actuators.

3.2 Instrumentation and Control Mechanisms

To use Autopilot, an application developer places sensors and actuators in the source code, specifying the program variables that sensors should monitor and the control points that can be modified by actuators. Configuration options also allow one to control performance data buffer sizes and management policies, transmission frequencies to external clients, and attached data reduction

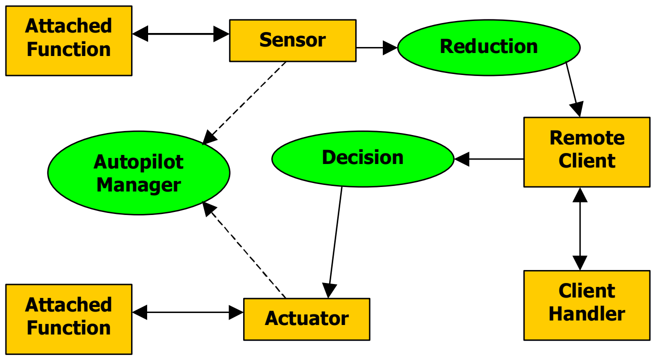

Figure 2 Autopilot Adaptive Control Toolkit Structure

functions to be applied before sensor data recording.

During execution, sensors and actuators can operate in either procedural or threaded mode. In procedural mode, they are invoked each time the application control flow reaches the relevant sensor or actuator. The advantage of this approach is that the granularity of measurement and control is determined solely by application behavior. However, the application can respond to changing environmental conditions only when control flow reaches an actuator.

Conversely, in threaded mode, the sensor or actuator executes in a separate thread, passively monitoring or changing application variables. Although this approach decouples monitoring or control frequency from control flow, it sacrifices precise synchronization with program state changes.

The functionality of Autopilot sensors and actuators can be extended via attached functions that process raw sensor data before it is recorded (e.g., to compute sliding window averages from data samples) or act on actuator commands using local data (e.g., to translate a command to increase a cache size into a specific expansion increment). This extension interface also enables derivation of qualitative behaviors from quantitative data – we have developed hidden Markov models and neural networks to classify I/O access patterns using request sizes and byte offsets.

Finally, external clients can dynamically attach to sensors and actuators based on tag matching. When sensors and actuators are created, they register with the Autopilot manager, specifying a set of tags. Clients can query the manager with a tag set, retrieving Globus pointers to any sensors or actuators with matching tags. With the pointers, the clients can attach to the sensors and actuators without knowledge of network location and either begin extracting performance data or exercising control.

3.3 Flexible Decision Procedures

Many control systems intertwine control mechanisms and policies. Although this may simplify design of a domain-specific control system, it makes it difficult to retarget the controller to a new environment or resource domain. Autopilot decouples resource policy decisions from the sensor/actuator mechanisms for control, allowing one to change policies without change to the policy mechanisms.

Standard implementations of policy decision procedures are often either algorithmic or conditionals or based on decision tables (e.g., using ideas from classic control theory). One of the lessons from parallel and distributed system performance analysis [7,8] is that performance optimization goals often conflict (e.g., the desire for high throughput and low response time). Moreover, the space of effective operating points for computational grid applications and their

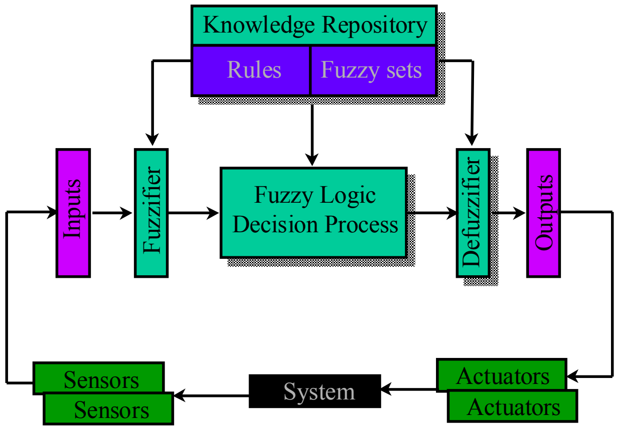

Figure 3 Autopilot Decision Process

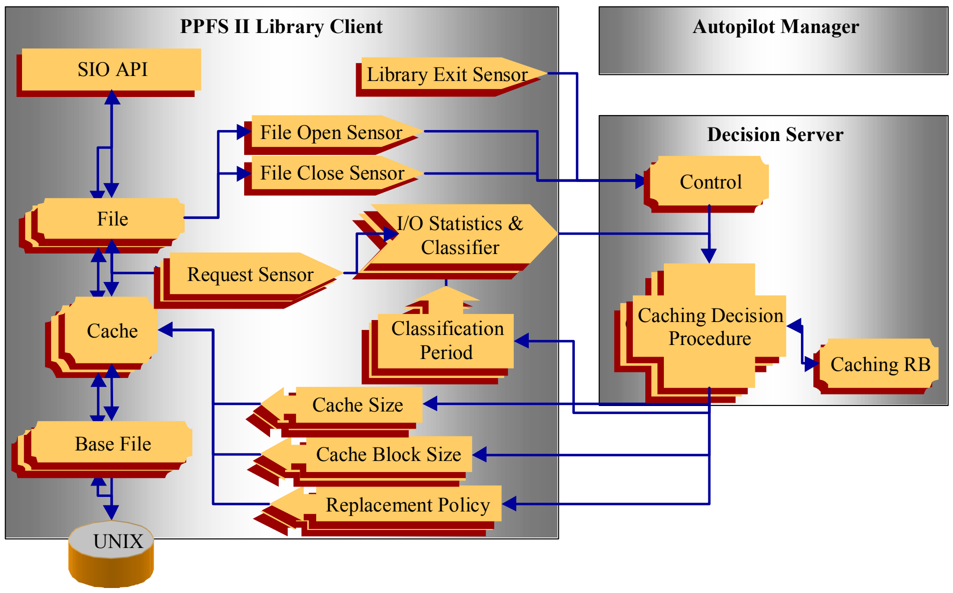

Figure 4 PPFS II Grid I/O Architecture

resources is poorly understood, enormously complex, and highly non-convex. Hence, it is impossible to describe a compact set of rules that will guarantee high performance.

To resolve this delimma, Autopilot includes a fuzzy logic engine that accepts sensor inputs and a resource management rule base and produces actuator controls. The attraction of fuzzy logic is its support for potentially conflicting rules, each of which can be partially true. By weighting these conflicting rules based on their relative truth, fuzzy logic can generate policies that smoothly balance conflicting goals [12].

As illustrated in Figure 3, Autopilot sensors capture quantitative and qualitative performance data that is fuzzified – converted to fuzzy sets. The fuzzy logic engine uses the rule base to evaluate this transformed data. The decision process' defuzzifier then transforms the fuzzy set outputs of all active rules into values for actuator control. When the actuators receive these requests, they change the target parameters within the system.

3.4 Standard Performance Daemons and Portable Interfaces

Autopilot also provides standard performance daemons that capture network and operating system performance data on distributed hosts. Typical data include processor utilization, disk activity, context switches, interrupts, memory utilization, paging activity, and network latencies. Using Autopilot's attribute-based registration mechanism, the daemons register with the Autopilot manger, enabling remote clients and decision processes to acquire data simply by attaching to the relevant daemon.

Finally, to enable "anywhere, anytime" performance analysis, Autopilot includes a portable Java interface for sensor data display and actuator control. This interface, called Autodriver, allows users to query the Autopilot manager for sensors, display selected sensor data using simple Java graphics, and inject changes with actuators using standard graphical widgets.

3.5 PPFS II: An Autopilot Case Study

High-performance computing has produced a variety of potential I/O configurations (e.g., disk data-striping factors, file and prefetching policies, and caching policies). Developing models for configuring parallel systems and distributed grid applications to effectively manage I/O resources and achieve good performance requires an exploration of this range of I/O configurations.

As a test of Autopilot functionality, as well as to address a difficult resource management problem, we have developed PPFS II, an adaptive, intelligent portable parallel file system. PPFSII relies on Autopilot sensors for real-time performance measurement, a fuzzy logic rule base for dynamic matching of I/O policies to application access patterns, and actuators to realize policy decisions. As illustrated in Figure 4, PPFS II is a user-level library built atop Autopilot that adapts file caching policies, relying on the underlying system software for physical I/O.

As a quantitative basis for adaptive performance optimization, instrumentation sensors capture salient aspects of both the application stimuli and system responses. Complementing quantitative performance data, knowledge of qualitative application behavior (e.g., sequential file access or latency-dominated communication) is obtained from automatic I/O pattern classification techniques [7]. The PPFS II fuzzy logic rule base supports adaptive striping (i.e., file distribution across variable numbers of disks) based on I/O contention and redundant storage to create multiple file representations that can be accessed efficiently by differing application access patterns.

Our experiences with PPFS II have shown that it is possible to achieve major performance improvements by adaptively matching access patterns to available resources and caching, prefetching, and file placement policies [7, 9]. Moreover, the Autopilot fuzzy logic rule base has allowed us to balance the desire for low latency access when there are only a small number of competing I/O streams against the need for high throughput when large numbers of tasks read or write files.

4. Virtue: Performance Immersion and Interactive Steering

For application developers and performance analysts to optimize the behavior of grid applications, they must understand the interactions of tens of distributed parallel systems connected by high-bandwidth networks, all accessing distributed secondary and tertiary storage systems, and containing thousands of processors. Although Autopilot and other measurement tools for

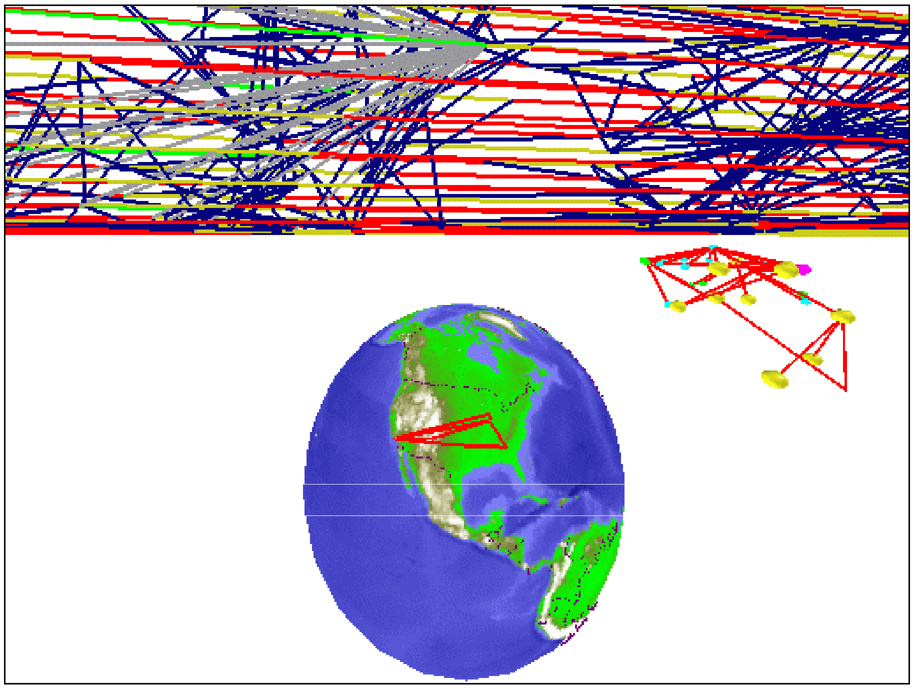

Figure 5 Virtue Hierarchical Views of Distributed Computations

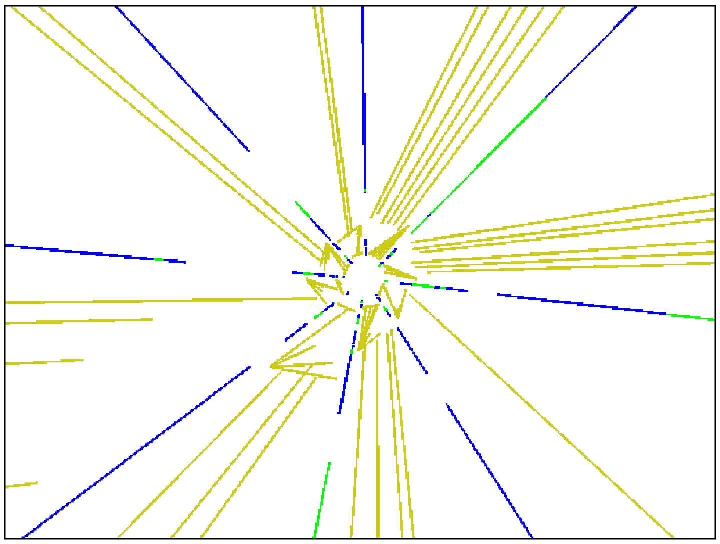
Figure 6 Virtue Time Tunnel View

computational grids provide the requisite performance data, the volume of data can be very large. This data must either be reduced or transformed to improve human comprehension.

We believe new performance visualization systems must more fully exploit human sensory capabilities and that they must support real-time display and analysis of codes as they execute. We are constructing a new, collaborative virtual environment, called Virtue [7,8], which is designed to eliminate the barrier separating the real world of users from the abstract world of software and its dynamics by making large, complex software and its behavior tangible entities that can be understood and manipulated in the same ways as the physical environment.

4.1 Toolkit Structure

The Virtue toolkit includes the following components

- Hierarchical graphs that show software structure, dynamics, and performance.
- Software component controls for direct manipulation of software behavior and structure.
- Manipulation tools for augmented interactions within the virtual environment.
- Multimedia annotation tools for distributed, collaborative exploration and recording.
- Standard external interfaces to Autopilot performance sensors and actuators for capturing software behavior and instantiating software changes.

The result is a high modality performance display and analysis system that allows users to interactively “drill down” from high-level geographic views of network traffic to low-level task or procedure dynamics.

4.2 Visualization Metaphors

Virtue and Autopilot work together to allow users to monitor and steer distributed computations in real time. Given the diversity and varying resolutions of real-time data, the underlying data collection system must intelligently capture only the data necessary for the current visualization. As users change visualization focus, Autopilot also changes the focus of its performance data collection.

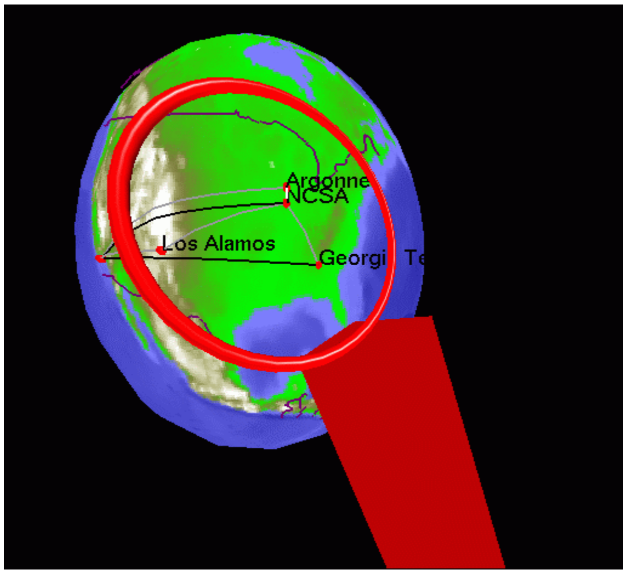

Figure 7 Virtue Magic Lens

In the context of computational grids, Virtue defines a visualization hierarchy for managing performance detail. As illustrated in Figure 5, at the highest level, users can view communication among the geographically distributed sites of an application's execution. Using data from the Autopilot daemons described in §3.4, users can judge communication network latencies, site utilization, and network routing.

Given a high-level perspective, users can select notes at multiple sites to "drill down" for more detail on the computation at those sites. At this lower level, users can interact with a time tunnel metaphor [4], illustrated in Figure 6, that shows the evolutionary behavior of all tasks at the selected site(s). The time tunnel represents task behavior as time lines along the periphery of a cylinder, with each line segment colored to denote the dominant activity during the associated interval. Chords through the interior of the time tunnel cylinder represent messages flowing between interacting tasks.

From the time tunnel's aggregate view of task behavior at a particular site, one can glean insights about communication and computation delays. From this insight, one can drill down further by selecting a single task for more detailed analysis. This final level displays a call graph of the executing procedures, with call graph vertices color-coded to show performance metrics.

The Virtue 3-D call graph display allows users to map a variety of statistics to each vertex and edge. In Figure 5, the color of each call graph vertex is bound to relative execution time during a sliding window. In this mapping, procedures that consume large amounts of processor time are shown as red, whereas those that consume little time are blue. However, Virtue supports a much richer set of mappings, allowing call graph attributes to be bound to invocation counts and hardware performance metrics (e.g., cache misses or floating point operations).

4.3 Direct Manipulation Tools

Visualization is but one aspect of optimization – it provides the insights needed to tune grid codes but not the mechanisms. For this reason, Virtue includes a toolkit for directly manipulating visual objects.   As suggested by Figure 7, these tools allow users to display additional information about graphs or calculate a derived metric.

In most cases, a 3-D visualization shows only a portion of the information associated with high-dimensional performance data. Virtue's generalized magnifying glass, called a magic lens, allows users to uncover this additional information by interactively focusing a tool on a portion of the visualization. For example, in a call graph visualization, the magic lens can expose hardware counter data, procedure names, or invocation counts when held above a call graph vertex.

Virtue's cutting plane enables users to slice a Virtue graph and compute metrics on the partitioned components. For example, if the cutting plane bisects a group of edges, the plane may compute the minimum, maximum, or average values mapped to the edges. In a time tunnel, these metrics correspond to communication volume or counts.

Finally, a set of actualization interfaces allow users to directly manipulate visualizations – grasping and adjusting glyphs to change software behavior (e.g., changing cache size by moving a cache size slider). These actualization interfaces (e.g., three-dimensional sliders attached to graph vertices) are bound to Autopilot actuators. When a user changes the setting of the actualization interface, the change is injected into the executing system via Autopilot actuators.

5. Related Work

Software mediated dynamic adaptation has been applied in many domains, including real-time and fault-tolerant systems, dynamic load balancing, on-line configuration management and adaptive input/output systems [11]. The Autopilot toolkit differs by emphasizing portable performance steering and closed-loop adaptive control and by decoupling the steering infrastructure from the policy domain.

Likewise, a plethora of techniques for distributed decision making have been proposed, ranging from decision tables and trees through standard control theory to fuzzy logic. Although each has strengths, fuzzy logic targets precisely the attributes of the performance optimization problem that challenge classic techniques, namely conflicting goals and poorly understood optimization spaces [12]. Autopilot builds on this observation by coupling a configurable fuzzy logic rule base for distributed decision making with wide area performance sensors and policy control actuators.

Virtue shares many attributes with emerging information visualization systems, though its focus on performance data immersion, real-time display, and direct manipulation for grid control is unique. The notion of virtual tools, including magic lenses, is rooted in work by Bier *et al* [2].

6. Conclusions

The combination of Autopilot's distributed, real-time measurement infrastructure and Virtue's immersive environment is an important step in creating a flexible software toolkit for grid application development, analysis, and tuning. Using Autopilot data, Virtue users can interactively explore multiple levels of detail, drilling down from geographic scale visualizations to the behavior of code in a single procedure.

Despite the promise of Autopilot and Virtue, much work remains. In particular, the Virtue actualization interfaces are incomplete, with tighter coupling to remote applications required. Moreover, a richer set of direct manipulation tools is needed, and we must enable drilldown from procedure call graphs to actual source code.

To enhance the Autopilot toolkit, we plan to develop a new set of attached functions for local data reduction, expand the sensor suite by fully integrating our I/O characterization software, and develop new fuzzy logic rule bases for network management and distributed task scheduling.

Finally, we are continuing to test the integrated Autopilot/Virtue toolkit with emerging grid applications.

Acknowledgments

Virtue and Autopilot are the work of a dedicated team of researchers in the Pablo group: Ruth Aydt, Mario Medina, James Oly, Mario Pantano, Ben Schaeffer, Donald Schmidt, Eric Shaffer, Huseyin Simitci, Nancy Tran, and Shannon Whitmore.

References

1. V. S. Adve, J. Mellor-Crummey, M. Anderson, K. Kennedy, J.-C. Wang, and D. A. Reed, "An Integrated Compilation and Performance Analysis Environment for Data Parallel Programs," *Proceedings of Supercomputing '95*, December 1995.
2. E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, “Toolglass and Magic Lenses: The See Through Interface, *Proceedings of ACM SIGGRAPH '93*, pp. 73-80, August 1993.
3. I. Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit,” *International Journal of Supercomputer Applications*, 11(2), pp. 115-128, 1997.
4. D. A. Reed, K. A. Shields, L. F. Tavera, W. H. Scullin, and C. L. Elford, “Virtual Reality and Parallel Systems Performance Analysis,” *IEEE Computer*, November 1995, pp. 57-67.
5. D. A. Reed, “Experimental Performance Analysis of Parallel Systems: Techniques and Open Problems,” *Proceedings of the 7th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, May 1994, pp. 25-51.
6. D. A. Reed, “Performance Instrumentation Techniques for Parallel Systems,” *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.
7. Reed, D., A., Elford, C. L., Madhyastha, T., Smirni, E., and Lamm, S. L., “The Next Frontier: Closed Loop and Interactive Performance Steering*,” Proceedings of the International Conference on Parallel Processing Workshop*, pp. 20-31, August 1996.
8. D. A. Reed, R. A. Aydt, L. DeRose, C. L. Mendes, R. L. Ribler, E. Shaffer, H. Simitci, J. S. Vetter, D. R. Wells, S. Whitmore, and Y. Zhang, "Performance Analysis of Parallel Systems: Approaches and Open Problems," *Proceedings of the Joint Symposium on Parallel Processing (JSPP)*, pp. 239-256, June 1998.
9. R. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, “Autopilot: Adaptive Control of Distributed Applications,” *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC),* 1998, pp. 172-179.
10. Smirni, E., and Reed, D. A., “Workload Characterization of Input/Output Intensive Parallel Applications,” *Proceedings of the Ninth International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, June 1997
11. J. S. Vetter, “Computational Steering Annotated Bibliography,” *SIGPLAN Notices*, 32(6), pp. 40-44, 1997.
12. L. A. Zadeh, Fuzzy Sets, *Information and Control*, Vol. 8, No. 3, pp. 338-353, June 1965.