Orion : Looking for Constellations in Physical Memory†

Aniruddha Bohra, Arati Baliga, Liviu Iftode
Department of Computer Science, Rutgers University
110 Frelinghuysen Road, Piscataway, NJ-08854
{bohra,aratib,iftode}@cs.rutgers.edu

Despite making significant strides in system characterization using fine grained instrumentation or macroanalysis, it is still a challenging task to understand and predict the behaviour of a computer system. The thesis of this paper is that there is a correlation between memory modification patterns and the high-level behavior of the system. To explore this property, we propose a holistic observation of a system's memory using a non-intrusive approach for sampling and analysis. Our preliminary results have demonstrated that holistic memory observation is feasible and does indeed correlate with system's condition. We conclude with a discussion on future research directions potentially enabled by our findings.

I. Introduction

As the complexity of the computer systems increases, their behavior is becoming increasingly harder to understand and predict. Systematic approaches, like annotating and analysing the code paths work well for applications and subsystems whose behaviour is well defined. However, for large and complex system software, these approaches have limited effectiveness. In this paper, we argue for a *holistic approach* to understanding system behaviour based on physical memory observation. At first glance, translating physical memory changes into system understanding appears as irrational as the belief that constellations in the sky affect events and humans on Earth. Very preliminary results presented in this paper, surprisingly show the contrary.

The thesis of this paper is that the memory behavior of a computer system can be correlated with its high-level behavior. The basis of our position is that memory is used universally by all software and its contents can be observed externally (e.g. from PCI devices) without executing any software on the system under test. The major apprehension is related to the relevance of a holistic observation of a system's memory to understanding its behaviour. To address this doubt, we designed Orion, a monitoring system, which we used to explore two fundamental questions: *(i)* can we access the memory of a system *externally* and continuously monitor it without significant overhead? and *(ii)* can we infer high-level system properties by monitoring and correlating memory modifications, with or without knowledge of its logical content?

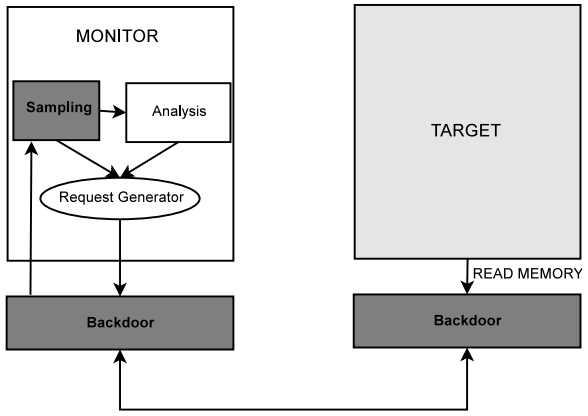

Fig. 1. Orion system architecture. The monitor treats the target as a black box. The Sampling Framework (dark shade) includes a Backdoor on both the target and the monitor.

The main contributions of this paper are: *(i)* a solution based on commodity hardware for physical memory monitoring, that operates continuously and independently of the OS, *(ii)* a monitoring methodology that identifies memory access patterns and helps infer high-level system properties. *(iii)* a preliminary set of data that supports interesting interpretation, and *(iv)* a discussion of opportunities, challenges and limitations of the holistic memory observations.

This paper is organized as follows. Section II describes the Orion architecture, the design considerations and our prototype implementation. Section III discusses preliminary results and outlines our approach to address the problems defined above. We outline future research directions in Section IV. A summary of related work is presented in V and we conclude in Section VI.

II. Orion Architecture

The key idea in Orion is to continuously sample the entire physical memory to identify the *variations* in behaviour. Our goal is to analyze deviations from the

†Please be patient, this paper takes a long time to print due to a large figure on Page 4.

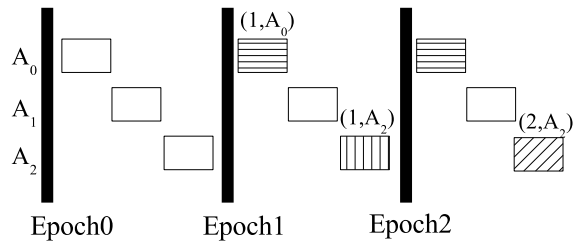

Fig. 2. Sampling and construction of a memory profile. Sampling is performed round robin over the region of interest identified by block addresses $A_0 \ldots A_2$. An epoch is defined as the period between two consecutive samples of the same block. The modified blocks are annotated with coordinates associated with the modification event in the memory profile. A point (x,y) indicates block y was modified during epoch x.

expected (correct) behaviour in *near-real time* due to significant load changes, system crashes, intrusions, etc. A summary of the modifications to a memory region over time is called a *profile*. Profiles are generated through continuous monitoring of reference systems, either under controlled load conditions or under normal operation. These profiles, called *reference profiles*, are then used to compare against the observed system memory variations.

Figure 1 shows the basic architecture of Orion. The system under test is shown on the right as the Target. The monitor, shown on the left, executes the Orion software.

A. Orion Components

The Data Sampling Framework (DSF) includes a Backdoor at both the monitor and the target. The Backdoor is a mechanism to enable direct access to the target system memory without involving the OS or executing any software on the target. A Backdoor can be implemented over an intelligent NIC with customized firmware or as a system component in a privileged domain on a Virtual Machine Monitor like Xen [1]. The request generator drives the data collection over the Backdoor and retrieves the samples. Each sample consists of a description of a memory region ($< address, length >$) and its contents.

The Data Analysis Framework (DAF) uses these samples to store a summary of the sample and compares the previous consecutive samples from the same memory block. It is also responsible to dynamically adjust the sampling granularity and frequency to effectively identify correlations between memory modification patterns.

Figure 2 shows the data sampling and analysis methodology used in Orion. It shows a region consisting of three blocks (A_0 to A_2). The blocks are sampled from the target memory over the Backdoor in a round robin fashion. The time between two consecutive samples for the same block is called an epoch. A summary of a block's contents is compared against the summary from its previous sample. The summary is created by using a collision resistant hash [2] and stored at the monitor. An event occurs when a block is modified between two consecutive samples. Figure 2 also shows the coordinates associated with modification events in the memory profile. In the profile, a point (x,y) implies that block y was modified in epoch x. The unmodified blocks (showing no coordinates) do not appear in the profile.

Sampling Granularity and Frequency Tradeoff. Sampling granularity refers to the maximum size of the region checked for modifications in each sample. Ideally, the system would sample each byte of the memory in every scan. However, the overhead of setting up such individual transfers increases linearly with the memory size. Moreover, continuous monitoring of the memory generates a large amount of data. The space requirements grow linearly with the number of individual regions we sample. At the same time, sampling the entire physical memory affects the frequency and the system cannot differentiate between fast changing memory regions and largely static ones. One solution is aggregating regions and sampling multiple regions at the same time. While it is an attractive solution for both dynamic frequency and granularity, by aggregating memory regions for sampling we may lose the localized modifications which allow us to pinpoint the affected regions. The goal of the monitoring mechanism is to *track* modifications. Intuitively, the sampling mechanism should sample the fast changing regions frequently while checking the largely static regions occasionally. In our current system, we sample memory regions idle for more than five epochs at the lowest frequency. On detecting a modification to such a region, the system starts sampling it at the maximum frequency.

B. Implementation

To build a prototype Orion system, we used two identical Dell Poweredge server systems running Linux 2.4.18 OS over a Pentium IV 2.8GHz processor, 1GB RAM, 1Gbps Ethernet NIC, and a 2Gbps Myrinet PCI-XD programmable network interface with 225MHz Lanai-X RISC processor and 2MB SRAM. We modified the firmware of the Myrinet NIC (Myrinet Control Program) to receive and interpret specialized Orion sampling requests. The monitor provides the address and length of the region of interest to the Backdoor on the target. The network processor on this Backdoor is used to initiate

the DMA from the system memory on receiving the sampling request. It is important to note that the target OS is not involved in this transfer.

The monitor performs two tasks on receiving a sampling reply. First, it creates a summary of the contents using a collision resistant secure hashing algorithm. Second, this hash is compared against the contents of the same region from the previous sample. An event is recorded if the hashes are not identical.

C. Limitations

The Backdoor is implemented external to the main processing path, as a PCI device or as a component of the privileged domain in a VMM. This leads to the Backdoor having a much slower access path to the memory compared to the host processor. Between two samples, the host processor can modify the memory contents multiple times. This makes it impossible to detect extremely fast modifications, that restore the original contents of a memory region *before* the next sample is retrieved. However, we believe that such modifications do not affect the overall system behaviour and focus on more *stable* modifications to the memory. To this end, we make a simplifying assumption that modifications between two samples are treated as a *single event*.

Another limitation of our system is the PCI bus access. While Orion does not use the target CPU or memory resources, the PCI bus is a shared resource and must be used by the Backdoor to access the memory of the system. This is an unavoidable overhead and can lead to some degradation in the system performance. However, our experiments with a similar system demonstrate that this degradation is negligible even at very high sampling rates [3].

III. Preliminary Results

In this section, we report preliminary results showing (i) that continuous memory monitoring through Orion is feasible and (ii) that the memory profile can be correlated with the system behavior. In all the experiments described in this section, we sample the entire 1GB memory of our test system in page sized (4KB) chunks. The methodology and reported results are conservative, as they are presented only as a proof of feasibility. Therefore, we do not reduce the overhead by pruning away regions of uninteresting or largely unmodified memory.

Epoch Length. An important question about Orion monitoring is the length of the epoch. If the epoch is too long, the monitoring is useless as it is unlikely to show any meaningful correlation between memory profiles and system properties. On the other hand, if the epoch is too small it adds unnecessary overhead on the monitor for both CPU cycles as well as space to store the samples. In our experiments, the epoch for sampling the entire memory in 4KB chunks, over the Backdoor described in Section II, is approximately one minute. While this epoch length may seem excessively long, we reiterate that these experiments are conservative and preliminary, as we do not expect to monitor the entire memory to infer interesting properties. For example, for the experiments described below, more than 99% of the memory was never modified in the idle system, and was modified less than three times under the severe load conditions described below.

Figure 3 shows the memory profiles generated using the methodology described in Section II. On the left is an idle system, while on the right is a system under load. The load is generated by executing a synthetic memory hog program that continuously spawns new processes. Each process thus created allocates memory in a tight loop This program allocates memory until the system memory and the swap space are exhausted and the Out Of Memory (OOM) handler of the OS proceeds to kill the offending processes.

For the idle system, we can observe a clear clustering of modified blocks with large areas of unmodified memory. The range of memory from 200MB to around 800MB is largely unchanged through 50 samples. However, an interesting modification pattern can be observed for some memory regions around 475MB and 740MB. Finally, the memory beyond 800MB changes continuously in all samples. We are currently investigating these patterns by zooming in the interesting regions for more detailed profiles and by identifying their logical content.

At first glance, the loaded system memory profile is much more dense compared to the idle system. A closer examination of these profiles in the context of the system behavior executing a memory-hog program reveals several interesting correlations: (i) The extensive memory changes during epochs 7-9 correspond to the execution of the memory-hog program. (ii) The patterns of memory modifications also track the execution of the system OOM handler. The system OOM handler executes in two stages: first, it terminates processes with the highest resident set size, and second it terminates all processes belonging to a user. These two stages of the OOM can be observed in the density of points between epochs 8 (first OOM stage begins) and 25 (second OOM stage solves the problem). After epoch 25, there are no more memory hog processes and the system goes back to the idle state.

These preliminary results are promising but, at the

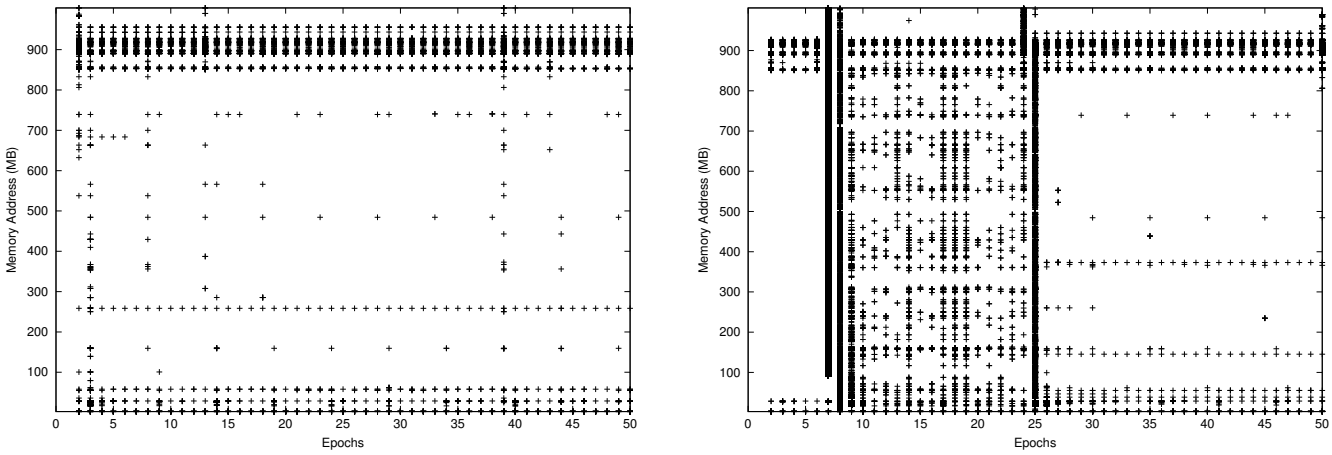

Fig. 3. Behaviour of the memory of an idle system (left), and under load (right). On the idle system there are large regions of memory that are unmodified. We observe some periodicity with gaps in modifications of addresses close to 500MB, while the regions between 800MB and 1GB are modified during all epochs. On the loaded system, a synthetic memory hog program that spawns several processes that execute a tight loop while allocating the maximum possible memory. The out of memory (OOM) handler is invoked by the OS when the swap space is exhausted. It initially kills individual processes, and during epoch 25 kills all processes for a user. The system then behaves very similar to an idle system.

same time, they lead to more questions and open problems. While it is true that significant memory activity indicates a loaded system, can we say when a system can recover and when it cannot? Can we identify a *signature* of memory usage which can identify an impending period of high load? What regions of the statically allocated OS memory are affected by such programs and are they important? Do all operating systems demonstrate a similar behaviour or is it OS specific? We plan to answer these and many similar questions in the near future.

IV. Future Research Directions

There are several interesting research directions that we plan to pursue in the near future. The ultimate research goal is to explore the potential of the holistic memory monitoring for system diagnosis. On the engineering side, the question is whether Orion can be implemented entirely on the local Backdoor . This eliminates the requirement of a remote monitor, but it requires efficient algorithms to reduce CPU and memory requirements.

Memory Monitoring as Data Streams. Orion memory monitoring has characteristics similar to applications processing massive data sets, e.g. monitoring IP network flows, sensor network databases, customer click streams, telephone records etc. The number of samples generated can be very large; the same samples cannot be retrieved again unless explicitly stored at the Monitor; the analysis does not require exact matches of content, instead we are interested in the *comparison* and *aggregation* of the sampled profiles. Recent theoretical work has demonstrated the use of a *data stream model* for such applications. A data stream is an ordered sequence of points that can be read once or a small number of times. The algorithms developed for this model operate on the *summary* of past data, reducing the memory requirements while losing some precision (a few percentage points) compared to the algorithms accessing complete data-sets [4], [5], [6], [7].

Data streaming algorithms for finding significant differences in network flows have demonstrated improvements in performance while significantly reducing memory requirements for processing the data-sets [8]. We are interested in finding similar differences in the memory profile of the target system. In Orion, we map the behavorial profile generated by continuous monitoring to a data stream model similar to that used in IP stream monitoring [8]. These algorithms are especially appealing as they process the data streams in near real-time (at router line speeds), can be implemented in hardware and use limited memory (SRAM) available on the programmable NICs.

Identification of System Structure. In this paper, we do not assume any knowledge of the system structure. However, information about the internal data structures can help our system to direct its continuous monitoring and identify invariants in their usage. It is important to note that there is no need for complete data definition, instead the additional information is used to define more fine-grained profiles of memory modification. A similar approach for storage systems has been previously used to identify live blocks across failures and in improving

the storage system performance [9].

V. Related Work

Monitoring global memory for performance over a shared virtual memory protocol was proposed in [10]. More recently, PCI devices were used in `Copilot` [11] for integrity verification and intrusion detection. The goal of `Copilot` was to continuously scan the memory for well known signatures of rootkits to detect intrusion.

Recently proposed commercially available remote management consoles [12], [13] are privileged intelligent PCI devices similar to Backdoor . These devices can be accessed remotely even when the host system is not available due to a hang or a crash. An industrial standard for remote communication with these devices over ethernet or serial interfaces has also been proposed [14]. However, apart from the remote console, these devices are used only to gather information from various environmental sensors in the chassis of the system.

Monitoring a system externally to infer code-paths and dependencies for application level software has been demonstrated to be efficient and effective [15]. Defensive Programming [16] advocates annotating software with monitoring counters and sensors to modify the execution behaviour of the program. This approach is extended in [17], [18] for exposing the codepaths within the OS and the interaction between the application and the OS. In the Gray Box systems approach, system properties are inferred by observing the behaviour of specially crafted requests and studying their responses [19]. This approach has been applied to storage and network subsystems to improve performance and reliability [20], [21].

External monitoring using request tracking for performance debugging and for achieving service level objectives has recently been proposed [22], [23]. In [24], [25], the system is monitored externally using predefined performance monitoring counters provided by the OS or applications and their variations correlated to the service level objectives.

In contrast to the above, in Orion we propose a holistic approach to monitoring and diagnosis which does not rely on a previously known protocol or application characteristics. Moreover, we do not assume that we have access to the source code and we do not execute any code on the system being monitored. Orion has the goal of identifying correlations between modifications to regions of memory and high-level system properties and behavior.

VI. Conclusions

Despite making significant strides in trying to infer system behaviour through fine grained instrumentation or macroanalysis, it is still a challenging task to understand the behaviour of the entire system including the OS. We propose a holistic approach of monitoring memory to infer high-level system properties from memory modification patterns.

We have presented Orion, a non-intrusive system architecture that continuously samples the memory, summarizes it, and uses these summaries to infer high-level system properties. Our initial results have shown that holistic memory observation is feasible, significant and interesting. We plan to extend this work in two main directions, *(i)* develop a diagnosis model for holistic memory observation by identifying correlations between memory modification properties and system behaviour, and *(ii)* explore efficient algorithms for sampling and analysis in order to reduce the resource requirements enough, to implement Orion entirely on the Backdoor.

References

[1] Paul Barham et al., "Xen and the Art of Virtualization," in *Proc. of the 19th ACM Symposium on Operating Systems Principles*, 2003, pp. 164–177.

[2] "Secure Hash Standard," Federal Information Processing Standards Publication, April 1995, FIPS PUB 180-1.

[3] F. Sultan et al., "Nonintrusive Failure Detection and Recovery for Internet Services using Backdoors. ," Tech. Rep. DCS-TR-524, Rutgers University, Dec. 2003.

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom, "Models and Issues in Data Stream Systems," in *PODS '02: Proc. of the* 21^{st} *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 2002, pp. 1–16, ACM Press.

[5] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan, "Clustering Data Streams: Theory and Practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.

[6] Graham Cormode and S. Muthukrishnan, "What's hot and what's not: tracking most frequent items dynamically," in *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2003, pp. 296–306.

[7] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan, "Comparing Data Streams Using Hamming Norms (How to Zero In)," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 529–540, 2003.

[8] Graham Cormode and S. Muthukrishnan, "What is new: Finding significant differences in network data streams," in *Proc. of the* 23^{rd} *Conference of the IEEE Communications Society, INFOCOM 2004*, Hong Kong, March 2004.

[9] Muthian Sivathanu, Lakshmi Bairavasundaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, "Life or Death at Block-Level," in *Proc of the* 6^{th} *Symposium on Operating Systems Design and Implementation (OSDI '04)*, San Francisco, CA, December 2004, pp. 379–394.

[10] Cheng Liao, Dongming Jiang, Liviu Iftode, Margaret Martonosi, and Douglas W. Clark, "Monitoring Shared Virtual Memory on a Myrinet Based PC Cluster," in *Proc. International Conference on Supercomputing*, July 1998, pp. 251–258.

[11] N.Petroni et al., "Copilot: a coprocessor-based kernel runtime integrity monitor," in *Proc. 13th Usenix Security Symposium*, San Diego, CA, aug 2004, pp. 179–194.

[12] "IBM Remote Supervisor Adapter II - Overview," http://www-307.ibm.com/pc/support/site.wss/document.do?lndocid=MIGR-50116.

[13] HP, "Remote Insight Lights-Out Edition II," http://h18004.www1.hp.com/products/servers/management/riloe2/.

[14] "Intelligent Platform Management Interface Specification v 2.0," Intel Developer Forum, http://www.intel.com/design/servers/ipmi/spec.htm, Sep 2004.

[15] Mike Chen, Anthony Accardi, Emre Kiciman, Jim Lloyd, David Patterson, Armando Fox, and Eric Brewer, "Path-Based Failure and Evolution Management," in *Proc. 1st USENIX / ACM Symposium on Networked Systems Design and Implementation (NSDI '04).*, San Francisco, CA, March 2004, pp. 309–322.

[16] Xiaohu Qie, Ruoming Pang, and Larry Peterson, "Defensive Programming: Using an Annotation Toolkit to Build Dos-Resistant Software," in *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[17] Yaoping Ruan and Vivek Pai, "Making the "Box" Transparent: System Call Performance as a First-Class Result," in *Proc. Usenix Annual Technical Conference*, Boston, MA, June 2004, pp. 1–14.

[18] Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal, "Dynamic Instrumentation of Production Systems," in *Proc. Usenix Annual Technical Conference*, Boston, MA, June 2004, pp. 15–28.

[19] Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Nathan C. Burnett, Timothy E. Denehy, Thomas J. Engle, Haryadi S. Gunawi, James Nugent, and Florentina I. Popovici, "Transforming Policies into Mechanisms with Infokernel," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing (Lake George), New York, October 2003, pp. 90–105.

[20] Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, "Deploying Safe User-Level Network Services with icTCP," in *Proc. of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, San Francisco, CA, December 2004, pp. 317–332.

[21] Muthian Sivathanu, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, "Improving Storage System Availability with D-GRAID," in *Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST '04)*, San Francisco, CA, March 2004, pp. 15–30.

[22] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier, "Using Magpie for Request Extraction and Workload Modelling," in *Proc. 6th USENIX / ACM Symposium on Operating Systems Design and Implementation (OSDI '04).*, San Francisco, CA, December 2004, pp. 259 – 272.

[23] Marcos Kawazoe Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes.," in *Proc. 19th Symposium on Operating Systems Principles SOSP'03*, Bolton Landing, NY, October 2003, pp. 74–89.

[24] Michael Littman, Nishkam Ravi, Eitan Fenson, and Rich Howard, "An Instance-Based State Representation for Network Repair," in *Proc. 19th National Conference on Artificial Intelligence(AAAI '04).*, 2004, pp. 287 – 292.

[25] Ira Cohen, Jeffrey S. Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. of the 6th Symposium of Operating Systems Design and Implementation (OSDI)*, Dec 2004.