USENIX Association

Proceedings of HotOS IX: The 9th Workshop on Hot Topics in Operating Systems

Lihue, Hawaii, USA
May 18–21, 2003

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Magpie: online modelling and performance-aware systems

Paul Barham, Rebecca Isaacs, Richard Mortier, and Dushyanth Narayanan
Microsoft Research Ltd., Cambridge, UK.

Abstract

Understanding the performance of distributed systems requires correlation of thousands of interactions between numerous components — a task best left to a computer. Today's systems provide voluminous traces from each component but do not synthesise the data into concise models of system performance.

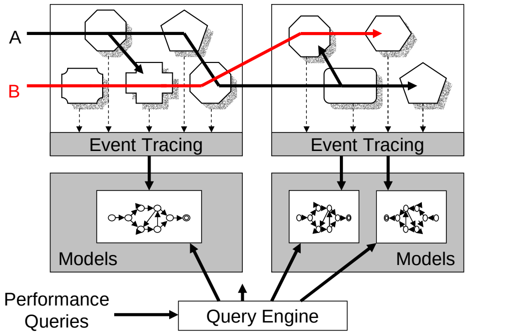
We argue that online performance modelling should be a ubiquitous operating system service and outline several uses including performance debugging, capacity planning, system tuning and anomaly detection. We describe the Magpie modelling service which collates detailed traces from multiple machines in an e-commerce site, extracts request-specific audit trails, and constructs probabilistic models of request behaviour. A feasibility study evaluates the approach using an offline demonstrator. Results show that the approach is promising, but that there are many challenges to building a truly ubiquitious, online modelling infrastructure.

1 Introduction

Computing today is critically dependent on distributed infrastructure. E-mail, file access, and web browsing require the interaction of many machines and software modules. When end-users of such systems experience poor performance it can be extremely difficult to find the cause. Worse, problems are often intermittent or affect only a small subset of users and transactions — the 'it works for me' syndrome.

Aggregate statistics are insufficient to diagnose such problems: the system as a whole might perform quite well, yet individual users see poor performance. Accurate diagnosis requires a detailed audit trail of each request and a model of *normal* request behaviour. Comparing observed behaviour against the model allows identification of anomalous requests and malfunctioning system components.

We believe that providing such models should be a basic operating system service. Performance traces should be routinely collected by all machines at all times and used to generate system performance models which are then made available for online programmatic query. To this end, we are building *Magpie*, an online modelling infrastructure. Magpie is based on two key design principles. *Black-box instrumentation* requires no source code modification to the measured system. *End-to-end tracing* tracks not just aggregate statistics but each individual request's path through the system.



The diagram shows how requests move through different software components across multiple machines in a distributed system. Magpie synthesizes event traces from each machine into models that can be queried programatically.

Figure 1. Magpie architecture.

Fine-grained, low-overhead tracing already exists for Linux [27] and Microsoft's .Net Server [17]; the challenge is to efficiently process this wealth of tracing information for improved system reliability, manageability and performance. Our goal is a system that collects fine-grained traces from all software components; combines these traces across multiple machines; attributes trace events and resource usage to the initiating request; uses machine learning to build a probabilistic model of request behaviour; and compares individual requests against this model to detect anomalies. Figure 1 shows our envisioned high-level design.

In Sections 2 and 3 we describe the many potential uses of online monitoring and modelling. Section 4 describes the current Magpie prototype, which does online monitoring but offline modelling. Sections 5 and 6 briefly describe related work and summarize our position.

2 Scenario: performance debugging

Joe Bloggs logs into his favourite online bookstore, monongahela.com*, to buy 'The Art of Surfing' in preparation for an upcoming conference. Frustratingly, he cannot add this book to his shopping cart, though he can access both the book details and his shopping cart. His complaint to customer support is eventually picked up by Sysadmin Sue. However, Sue is perfectly able to order 'The Art of Surfing' from her machine, and finds nothing suspicious in the system's throughput or availability statistics: she can neither replicate the bug nor identify the faulty component.*

Consider the same website augmented with the Magpie online performance modelling infrastructure. Magpie maintains detailed logs of resource usage and combines them in real time to provide per-request audit trails. It knows the resource consumption of each request in each of several stages — parsing, generation of dynamic content, and database access — and can determine whether and where Joe's request is out of bounds with respect to the model of a correctly behaving request.

Using Magpie's modelling and visualization tools, Sue observes a cluster of similar requests which would not normally be present in the workload model. On closer examination, she sees that these requests spend a suspicious amount of time accessing the 'Book Prices' table, causing a time-out in the front-end server. This is the problem which is affecting Joe and the culprit is a misconfigured SQL Server. Sue could not replicate the bug because her requests were redirected by an IP address based load balancer to a different, correctly configured replica. Within minutes the offending machine is reconfigured and restarted, and Joe can order his book in good time for the conference.

This is just one performance debugging scenario, but there are many others. File access, browsing and e-mail in an Intranet may rely on Active Directory, authentication and DNS servers: slow response times could be caused by any combination of these components. Diagnosing such problems today requires expert manual intervention using tools such as `top`, `traceroute` and `tcpdump`.

3 Applications

Pervasive, online, end-to-end modelling has many uses apart from distributed performance debugging; here we list some of the most exciting ones. These applications are research goals rather than accomplished facts. Section 4 describes our first step towards these goals in the form of our offline modelling prototype.

Capacity planning. Performance prediction tools such as *Indy* [11] require workload models that include a detailed breakdown of resource consumption for each transaction type. Manual creation of such models is time consuming and difficult; Magpie's clustering algorithm (Section 4) automatically creates workload models from live system traces.

Tracking workload level shifts. Workloads can change qualitatively due to subtle changes in user behaviour or client software. For example, a new web interface might provide substring matching in addition to keyword search. To reconfigure the system appropriately, we must distinguish level shifts from the usual fluctuations in workload. Magpie could do so by comparing models of current and past workloads. This might also detect some types of denial-of-service attacks.

Detecting component failure. Failure of software or hardware components can degrade end-to-end performance in non-obvious ways. For example, a failed disk in a RAID array will slow down reads that miss in the buffer cache. By tracking each request through each component, Magpie can pinpoint suspiciously behaving components.

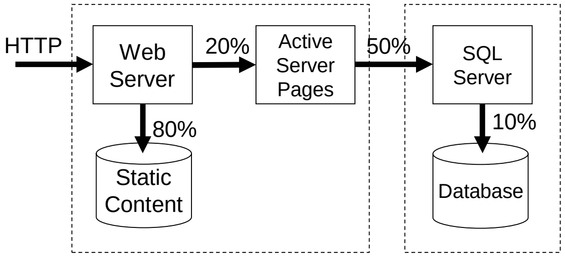
Comparison-based diagnosis. Workload models could be compared across site replicas to diagnose performance discrepancies.

'Bayesian Watchdogs'. Given a probabilistic model of normal request behaviour, we can maintain an estimate of the likelihood of any request as it moves through the system. Requests deemed unlikely can then be escorted into a safe sandboxed area.

Monitoring SLAs. The emerging Web Services standards [26] allow multiple sites to cooperate in servicing a single request. For example, an e-commerce site might access third-party services for authentication, payment, and shipping. Magpie performance models could be compared with service level agreements to check compliance. However, this does raise issues of trust and privacy of sensitive performance data.

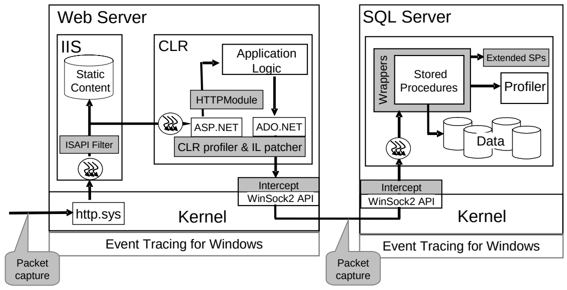
Kernel performance debugging. Response times on a single machine are governed by complex interactions between hardware, device drivers, the I/O manager, the memory system and inter-process communication [14]. Event Tracing for Windows already traces all these OS components, enabling request tracking at this fine granularity.

End-to-end latency tuning. Scalable self-tuning systems such as SEDA [25] optimise individual compo-



In a typical e-commerce site the majority of HTTP Requests are for static content, but a significant fraction require execution of code in the .Net runtime, which might issue SQL queries to a database server machine.

Figure 2. A simple e-commerce site.



Instrumentation points for the web server and database server in our test e-commerce site. Some components such as the http.sys kernel module and the IIS process generate events for request arrival, parsing, etc. Additional instrumentation inserted by Magpie (shown in gray) also generates events; all these events are logged by the Event Tracing for Windows subsystem.

Figure 3. Instrumentation points.

nents for throughput, with potentially deleterious effects on end-to-end latency. For example, they ignore concurrency within a single request, which has little effect on throughput. However, exploiting intra-request concurrency can reduce latency.

4 Feasibility study

To evaluate the feasibility of our approach, we need to answer the following three questions: Are the overheads of tracing acceptable? Is the analysis of log data tractable? Are the resulting models useful?

To this end, we have constructed an *offline* Magpie demonstrator, which traces in-kernel activity, RPCs, system calls, and network communication, using Event Tracing for Windows [17], the .Net Profiling API [18], Detours [12] and `tcpdump` respectively. We then ran an unmodified e-commerce application (Duwamish7) on the two-machine configuration depicted in Figure 2, and exercised it using a workload based on TPC-W [24]. Figure 3 shows the components and instrumentation points for this system.

Each instrumentation point generates a named event, timestamped with the local cycle counter. For example, we have events for context switches and I/O operations as well as entry into and exit from selected procedures. Offline processing assembles logs from multiple machines, associates events with requests, and computes the resource usage of each request between successive events. Figure 4 shows an automatically generated visualization of one such request audit trail.

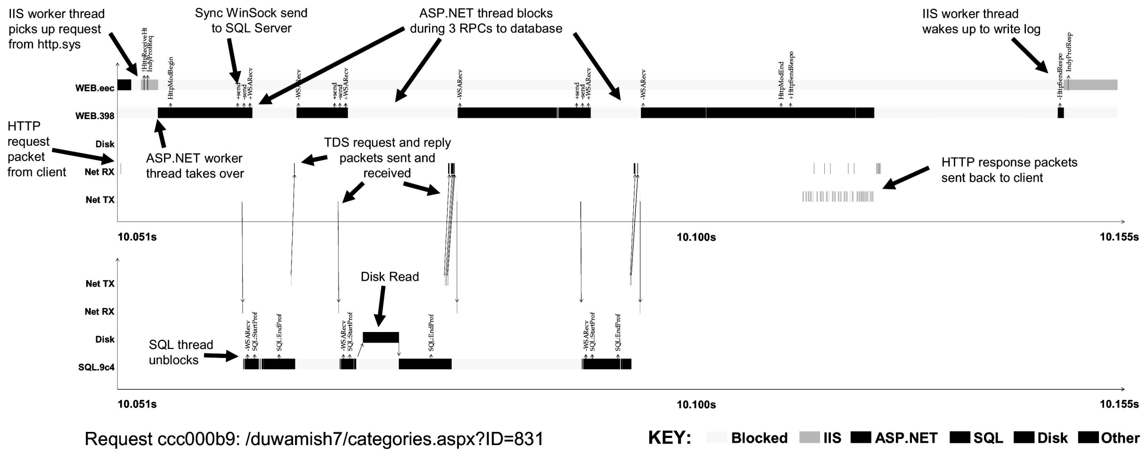
To stitch together a request's control flow across multiple machines, we use the logged network send and receive events. Similarly, we could track requests across multiple thread pools on the web server, by instrumenting the thread synchronization primitives. Our current prototype does not yet track thread synchronization: instead, we track a request across thread pools by adding a unique ID to the request header.

To estimate the worst-case overhead of logging, we ran a simple stress benchmark on our test site, and traced all system activity on both machines. This generated 150k events/min, resulting in 10 MB/min of log data. This large volume is mostly caused by inefficient ASCII logging of system calls, and could easily be reduced. On a simple microbenchmark, throughput was reduced by 18%, again with all trace points active.

Behavioural clustering

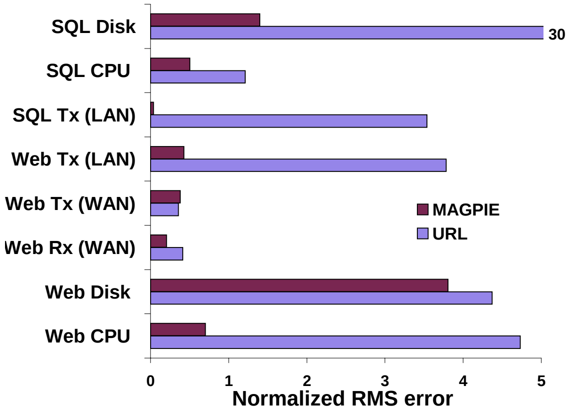
Our initial driving application was workload generation for the Indy performance prediction toolkit. This requires specification of a small number of *representative* transaction types together with their relative frequencies. Typically, these transaction types are categorized by their URL, and measured using microbenchmarks. However, URLs are not always a good indicator of request behaviour: the same URL often takes different paths through the system due to differences in session state or error conditions.

Instead, Magpie categorizes requests by their observed behaviour under realistic workloads. It merges logs and deterministically serializes each request's events to construct an *event string*, annotated with resource usage information. We then cluster the event strings according to the Levenshtein String Edit Distance [21] augmented with the normalized Euclidean distance between the resource usage vectors. Behavioural clustering gives us substantially better clustering accuracy than a URL-based approach, in terms of the difference between behaviour of the representative and those transactions it represents, as shown in Figure 5.


IIS worker thread picks up request from http.sys
Sync WinSock send to SQL Server
ASP.NET thread blocks during 3 RPCs to database
IIS worker thread wakes up to write log
WEB.eec
WEB.398
HTTP request packet from client
Disk
Net RX
Net TX
ASP.NET worker thread takes over
TDS request and reply packets sent and received
HTTP response packets sent back to client
10.051s
10.100s
10.155s
Net TX
Net RX
Disk
SQL.9c4
Disk Read
SQL thread unblocks
10.051s
10.100s
10.155s
Request ccc000b9: /duwamish7/categories.aspx?ID=831
KEY:
Blocked
IIS
ASP.NET
SQL
Disk
Other

An automatically generated timeline of a sample request across several resources (CPU threads, network, and disk) and different software components on two machines. The small vertical arrows represent events posted by instrumentation points and are labelled with the event name. Annotations explaining how the graph is interpreted have been added manually.

Figure 4. Request audit trail.



Comparison of two clustering methods applied to the same data, one based on Magpie clustering and the other on URL. A synthetic workload is generated using both sets of clusters and compared to the original dataset. This graph shows the RMS error of actual and predicted resource consumption for each model, broken down by resource type (errors are shown normalised by the mean).

Figure 5. Magpie vs. URL-based clusters.

Our unoptimized C♯ implementation takes just under 2 seconds to extract 8 clusters from 5 minutes of trace data (1800 requests) on a 2GHz Intel Pentium 4. This gives us confidence that an online version of Magpie modelling will have low overheads.

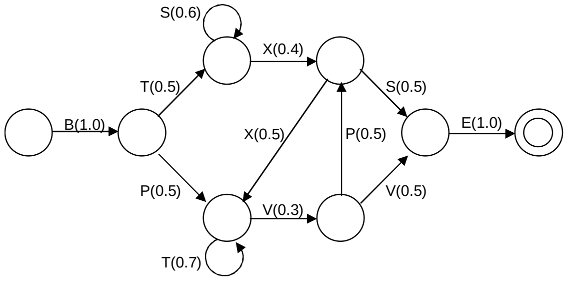
Inferring higher level behaviour

Behavioural clustering alone is sufficient for realistic workload generation, and additionally enables some aspects of the performance debugging scenario of Section 2. By looking for outliers — requests that are not close to any existing clusters — we can identify anomalous requests. However, this does not tell us the cause of the anomaly: the particular event or event sequence that makes the request suspicious.

To identify specific events that are out of place, we must model the process that generates the event sequences. A natural way to model such a process is as a probabilistic state machine: each transition between states produces an event, and has an associated probability. Given such a model and an anomalous request, we can identify events or event sequences with suspiciously low probability. In fact, an online version of such a model could be used to continuously check *all* requests for anomalous behaviour, i.e. to implement the "Bayesian watchdogs" scenario of Section 3.

A probabilistic state machine can be represented as a stochastic context-free grammar (SCFG). This grammar is an intuitive representation of the underlying process which generated the event sequences, and can provide useful clues as to the higher level control flow and the internal structure of the application code, including hierarchy and looping.

Given a set of example strings, the ALERGIA algorithm [4] derives an SCFG in linear time by recursively merging similar portions of their prefix tree. For the test grammar in Figure 6, our C# implementation converges to the correct state machine after 400 sample strings, taking around 30 milliseconds on a 2.5GHz machine. By way of comparison, the example 5 minute long dataset described earlier produced some 1800 event strings, implying that ALERGIA should have a convergence time of about a minute and negligible CPU overhead.



```
BPTTVPSE           BPVPSE         BPVPSE
BTSSSSSSSSSSSXSE   BPTVPXTTVVE    BPTTVVE
BTSXSE             BPVVE          BPVPSE
```

A stochastic context free grammar (the Reber grammar) and some of the strings which it generates. The string **BTSXSE**, for example, will be generated with probability 0.06.

Figure 6. Example SCFG.

This efficiency, combined with the enhanced information in the resulting model, has encouraged us to apply ALERGIA to Magpie request event strings. We are currently evaluating ALERGIA's performance on the longer and more complex strings generated by Magpie, and exploring extensions to the algorithm to incorporate resource usage measurements.

Modelling concurrency

Both the clustering and SCFG-based approaches use a deterministic serialization of each request's events, and thus do not model concurrency within a request. In our test scenario there was little intra-request concurrency; more complex systems will require us to explicitly capture concurrency, perhaps by extending the clustering distance metric and replacing SCFGs with coupled hidden Markov models [3]. In so doing, scenarios such as end-to-end latency tuning become feasible.

5 Related work

The closest relative to Magpie is Pinpoint [6], a prototype implementation of the *online system evolution* model [7]. Pinpoint has a similar philosophy and design to Magpie, recommending aggressive logging, analysis and anomaly detection, but its focus is fault detection rather than performance analysis. TIPME [8] performs continuous monitoring on a single machine, also for the purposes of debugging particular problems. In this case, the monitored environment encompasses the transactions initiated by user input to the X Windows system on a single machine.

Scout [19] and SEDA [25] require explicitly defined paths along which requests travel through the system. In contrast, Magpie infers paths by combining event logs generated by black-box instrumentation. Whole Path Profiling [16] traces program execution at the basic block level; Magpie's paths are at a much coarser granularity, but can span multiple machines.

Log-based performance profiling has been used in distributed systems [13], operating systems [23], and adaptive applications [20]. Magpie differs in that its logging is black-box and not confined to a single system. It also tracks resource usage of individual requests rather than aggregating information to a system component or resource.

Distributed event-based monitors and debuggers [1, 2, 15] track event sequences across machines, but do not monitor resource usage, which is essential for performance analysis. Systems such as that used by Appliant [5] measure web application response time by embedding JavaScript in the HTML of fetched pages which records the relevant data at the client browser. The aggregated data gives a view of server latency which would complement the detailed server-side workload characterisation obtained using Magpie.

Finally, a few model checking approaches infer correctness models from source code analysis [9] or runtime monitoring [10, 22]; this is similar to our approach of inferring performance models.

6 Conclusion

Our preliminary results show that fine-grained logging and offline analysis are feasible, and that the resulting models are useful for workload generation. Truly pervasive, online modelling has many potential applications, but also presents many challenges.

Not all events have information value for all applications. For example, we both instrument TCP sends and receives, *and* capture packets on the wire, redundant except when debugging TCP. Ideally, we would dynamically insert and remove instrumentation according to its utility.

Our modelling algorithms are offline, operating on a single merged log of all events. Online modelling will require incremental, distributed algorithms that can process events when and where they occur. We are currently developing the infrastructure to enable an online system. In addition, we need efficient ways to use the generated models: for example, a distributed database with an online query mechanism for performance debugging.

Although some of our target scenarios are ambitious given the current state-of-the-art, we believe that they will be achievable in the near future. Performance-aware systems are an important step towards automatic system management and an essential part of managing increasingly complex and distributed systems.

References

[1] *E. Al-Shaer, H. Abdel-Wahab, and K. Maly*. HiFi: A new monitoring architecture for distributed systems management. *Proc. IEEE 19th International Conference on Distributed Computing Systems (ICDCS'99)*, pages 171–178, May 1999.

[2] *P. C. Bates*. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Transactions on Computer Systems (TOCS)*, 13(1):1–31, 1995.

[3] *M. Brand, N. Oliver, and A. Pentland*. Coupled hidden Markov models for complex action recognition. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 994–999, June 1997.

[4] *R. Carrasco and J. Oncina*. Learning stochastic regular grammars by means of a state merging method. *Grammatical Inference and Applications*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 139–152. Springer-Verlag, 1994.

[5] *J. B. Chen and M. Perkowitz*. Using end-user latency to manage internet infrastructure. *Proc. 2nd Workshop on Industrial Experiences with Systems Software WIESS'02*, Dec. 2002.

[6] *M. Chen, E. Kiciman, E. Fratkin, E. Brewer, and A. Fox*. Pinpoint: Problem determination in large, dynamic, Internet services. *Proc. International Conference on Dependable Systems and Networks (IPDS Track)*, pages 595–604, June 2002.

[7] *M. Y. Chen, E. Kiciman, and E. Brewer*. An online evolutionary approach to developing Internet services. *Proc. 10th SIGOPS European Workshop*, Sept. 2002.

[8] *Y. Endo and M. Seltzer*. Improving interactive performance using TIPME. *Proc. ACM SIGMETRICS*, June 2000.

[9] *D. Engler, D. Y. Chen, and A. Chou*. Bugs as inconsistent behavior: A general approach to inferring errors in systems code. *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 57–72, Oct. 2001.

[10] *M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin*. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):1–25, Feb. 2001.

[11] *J. C. Hardwick*. Modeling the performance of e-commerce sites. *Journal of Computer Resource Management*, 105:3–12, 2002.

[12] *G. C. Hunt and D. Brubacher*. Detours: Binary interception of Win32 functions. *Proc. 3rd USENIX Windows NT Symposium*, pages 135–144, July 1999.

[13] *G. C. Hunt and M. L. Scott*. The Coign automatic distributed partitioning system. *Proc. 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, pages 187–200, Feb. 1999.

[14] *M. B. Jones and J. Regehr*. The problems you're having may not be the problems you think you're having: Results from a latency study of Windows NT. *Proc. 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Mar. 1999.

[15] *J. Joyce, G. Lomow, K. Slind, and B. Unger*. Monitoring distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 5(2):121–150, 1987.

[16] *J. R. Larus*. Whole program paths. *Proc. ACM conference on Programming language design and implementation (SIGPLAN'99)*, pages 259–269, June 1999.

[17] *Microsoft Corp.* Event Tracing for Windows (ETW). **http://msdn.microsoft.com/library/en-us/perfmon/base/event_tracing.asp**, 2002.

[18] *Microsoft Corp.* .Net Profiling API. **http://msdn.microsoft.com/library/en-us/cpguide/html/cpcondebuggingprofiling.asp**, 2002.

[19] *D. Mosberger and L. L. Peterson*. Making paths explicit in the Scout operating system. *Proc. 2nd USENIX Symposium on Operating Systems Design and Implementation (OSDI'96)*, pages 153–168, Oct. 1996.

[20] *D. Narayanan, J. Flinn, and M. Satyanarayanan.* Using history to improve mobile application adaptation. *Proc. 3rd IEEE Workshop on Mobile Computing Systems and Applications*, pages 31–40, Dec. 2000.

[21] D. Sankoff and J. Kruskal, editors. *Time warps, string edits and macromolecules: the theory and practice of sequence comparison*, chapter 1. CLSI Publications, 1999.

[22] *S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson*. Eraser: A dynamic data race detector for multithreaded programs. *ACM Transactions on Computer Systems*, 15(4):391–411, 1997.

[23] *M. Seltzer and C. Small*. Self-monitoring and self-adapting operating systems. *Proc. 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, pages 124–129, May 1997.

[24] Transaction Processing Performance Council. *TPC Benchmark W (Web Commerce) Specification* . **http://www.tpc.org/tpcw/**.

[25] *M. Welsh, D. Culler, and E. Brewer*. SEDA: An architecture for well-conditioned, scalable Internet services. *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 230–243, Oct. 2001.

[26] *World Wide Web Consortium*. Web Services. **http://www.w3c.org/2002/ws/**, 2002.

[27] *K. Yaghmour and M. R. Dagenais*. Measuring and characterizing system behavior using kernel-level event logging. *Proc. USENIX Annual Technical Conference*, June 2000.