The Need for a Whole-System View of Performance

Matthias Hauswirth
University of Colorado at Boulder
Matthias.Hauswirth@colorado.edu

Peter F. Sweeney
IBM Thomas J. Watson Research Center
pfs@us.ibm.com

Amer Diwan
University of Colorado at Boulder
diwan@colorado.edu

Michael Hind
IBM Thomas J. Watson Research Center
hindm@us.ibm.com

ABSTRACT

Middleware usually runs on top of a powerful execution platform. Often that platform includes extensive runtime libraries, a virtual machine, an operating system, and modern complex hardware.

The understanding of middleware performance is complicated by intricate interactions between the application, the middleware, and the components of the underlying platform.

The functional interfaces of the components of such a platform are clearly defined. They range from the API specification, over the virtual machine specification, the system call interface, to the processor's instruction set architecture. The *functionality* of a system component can be understood by just looking at that component's functional specification.

The *performance* of a system component, on the other hand, can often not be understood in isolation. This is because performance interactions between components can be very intricate, non-linear, and involve unpredictable delays.

We propose temporal vertical profiling [1] as an approach for whole system performance understanding. Temporal vertical profiling captures the temporal system behavior on all layers, and of all components, of the system. A trace of values for performance metrics from system components is captured and analyzed to identify correlations between such metrics. Correlation is the first step in the identification of causal releationships between metrics and components. And these causal relationships are the fundamental building blocks for performance understanding.

1. INTRODUCTION

Modern applications often are built on top of middleware for distributed communication, such as CORBA Object Request Brokers (ORB) [6], Java Remote Method Invocation (RMI) [5], or the Simple Object Access Protocol (SOAP) [8], and they make use of component technologies such as Enterprise Java Beans (EJB) [4] or the CORBA Component Model (CCM) [7].

The overall performance of such an application is dependent on the performance of all layers in the execution stack of the system (Figure 1). To decide which middleware and component technology to use, the performance of different implementations can be compared by measuring key performance metrics (usually some form of latency and throughput) running the *same* application on the *same* system (all the way from the Java library to the hardware layer), just interchanging the different implementations of the middleware and component technology layer (Figure 1).

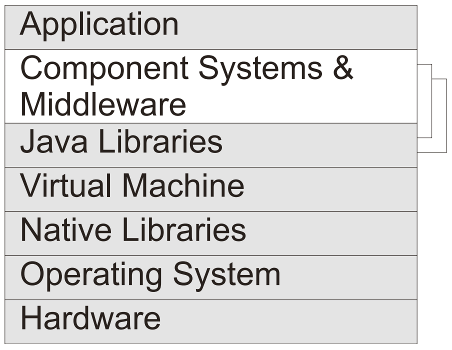

Figure 1: Layers of the Execution Stack.

But the overall system performance is more than the sum of its parts. It is influenced by various non-functional interactions between all these layers. A specific piece of application or middleware code can have bad performance because of reasons that are not directly caused by that code. Such reasons include dynamic optimization decisions in the runtime compiler, the memory management policy of the garbage collector, generated machine code leading to misspeculation in the processor, or interactions with other code executing in a different thread at the same time. To determine the overall performance of a system, it is sufficient to measure the key performance metrics. But to *understand* the reasons for good or bad system performance , one needs to study the performance interactions between the various layers of the system. In [1] we introduce *temporal vertical profiling* as an approach to understand such performance interactions between layers of the system.

2. TEMPORAL VERTICAL PROFILING

A computer system can be considered a state machine, with various kinds of events leading to different states. Each component in each layer emits different kinds of events, such as: instruction completions and cache misses in the processor, page faults and context switches in the operating systems, object allocations, garbage collections, and runtime compilations in the virtual machine, object serializations and distributed method invocations in the middleware, and business transaction completions and abortions in the application.

Temporal vertical profiling is an approach to whole-system performance understanding by discovering the causality relationships between the behavior of different layers. A whole-system event trace (the sequence of all events observed in all layers) is aggregated into a set of time series, one time series per performance metric of interest. Figure 2 shows a whole-system event trace (different height pins represent events happening on different system layers), where one event type (the red pins) corresponds to the aggregation interval boundaries (for each aggregation interval, there will be one data point in the time series). The figure shows two time series (for metrics m_1 and m_2) created from the event trace by counting two different types of events.

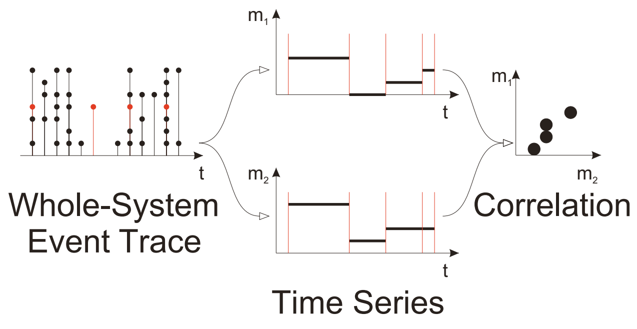

Figure 2: From Event Trace to Time Series.

Examples of performance metrics are IPC (instructions completed per cycle), object allocations per second, or transaction abortions per transaction completed. The time series are created by aggregating (counting) events over intervals. For example, all transaction completions and transaction abortions are counted for each scheduling time slice.

Figure 2 also shows a scatter plot of the two metrics, indicating that the two time series are correlatated. If two time series are temporally correlated, then there is a chance that one time series is influencing the other. For example, if the IPC gradually drops over time, and the cache miss rate gradually increases over time, we can conclude that there is a chance that the drop in IPC is caused by the increase in cache miss rate. This leads to the cause of the problem: if we want to improve the processor utilization (IPC), we should try to improve the cache performance.

We have successfully applied the temporal vertical profiling approach in [1], discovering different causes of bad processor utilization. All of these studies involved performance metrics on multiple layers of the system. This indicates that it is necessary to analyze all layers for an understanding of whole-system performance.

3. PERFORMANCE EXPLORER

The number of different metrics (and thus time series) to analyze can be large. Capturing all aspects of interest in a modern processor alone can already require several hundred metrics. Manually finding causality relations in such a large number of time series is very labor intensive, thus it is crucial to automate this process as much as possible.

We have developed Performance Explorer, a tool to visualize and analyze time series of performance metrics. Performance Explorer is a platform which can be extended to provide support for new types of data, new kinds of visualizations, and support for finding causality relationships. It provides a domain-specific language for filtering data and for computing metrics derived from other metrics (e.g. computing IPC given instructions completed and cycles). Performance Explorer uses a space-efficient self-defining data format, where the meta data of each trace defines the actual trace contents. This allows the use of Performance Explorer on traces that contain previously unknown event types. This means that the performance of new kinds of applications, middleware, component systems, or other layers providing previously unknown event types, can be analyzed without any changes to Performance Explorer.

We have successfully used Performance Explorer in [1] to explain unexpected performance phenomena of several Java benchmarks running on the Jikes RVM [3] virtual machine on AIX on a POWER4 [2] processor, and we are currently working on adding support for automating the detection of causality relationships where possible.

4. CONCLUSIONS

The execution stack of modern systems consists of many layers, where the middleware and component system are just one such layer. Understanding the performance of a system requires an understanding of the causality relationships between those layers. Temporal vertical profiling is a way to find such relationships and thus further the understanding of system performance.

5. REFERENCES

[1] Matthias Hauswirth, Peter F. Sweeney, Amer Diwan, and Michael Hind. Vertical profiling: Understanding the behavior of object-oriented applications. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. ACM Press, October 2004.

[2] IBM. Power4 system microarchitecture. http://www-1.ibm.com/servers/eserver/pseries/hardware/-whitepapers/power4.html, 2001.

[3] Jikes Research Virtual Machine (RVM). http://www.ibm.com/developerworks/oss/jikesrvm.

[4] Sun Microsystems. Enterprise JavaBeans specification, v2.1. http://java.sun.com/products/ejb/docs.html.

[5] Sun Microsystems. Java Remote Method Invocation (RMI). http://java.sun.com/j2se/1.4.2/docs/guide/rmi/.

[6] Object Management Group (OMG). Common Object Request Broker Architecture (CORBA/IIOP), v3.0.3. http://www.omg.org/docs/formal/04-03-12.pdf.

[7] Object Management Group (OMG). CORBA Components, v3.0. http://www.omg.org/docs/formal/02-06-65.pdf.

[8] W3C. Simple Object Access Protocol (SOAP). http://www.w3.org/TR/soap/.