Shedding Light on Distributed System Executions

Jenny Abrahamson
Facebook Inc.
Seattle, WA, USA
jennya@fb.com

Ivan Beschastnikh
U. British Columbia
Vancouver, BC, Canada
bestchai@cs.ubc.ca

Yuriy Brun
U. Massachusetts, Amherst
Amherst, MA, USA
brun@cs.umass.edu

Michael D. Ernst
U. Washington
Seattle, WA, USA
mernst@cs.washington.edu

ABSTRACT

In a distributed system, the hosts execute concurrently, generating asynchronous logs that are challenging to comprehend. We present two tools: ShiVector to transparently add vector timestamps to distributed system logs, and ShiViz to help developers understand distributed system logs by visualizing them as space-time diagrams. ShiVector is the first tool to offer automated vector timestamp instrumentation without modifying source code. The vector-timestamped logs capture partial ordering information, useful for analysis and comprehension. ShiViz space-time diagrams are simple to understand and interactive — the user can explore the log through the visualization to understand complex system behavior. We applied ShiVector and ShiViz to two systems and found that they aid developers in understanding and debugging.

Categories and Subject Descriptors: D.1.3 [Concurrent Programming]: Distributed programming
General Terms: Concurrency, Modeling, Visualization
Keywords: Log analysis, distributed systems

1. INTRODUCTION

Understanding and debugging distributed systems is challenging. It is difficult to reason about concurrent executions, reproduce race conditions, and understand how components communicate. Faced with these challenges, developers often log system executions. Logs are easy to create, and they allow developers to record useful runtime state and behavior information. However, manually inspecting logs is both tedious and complicated: logs can be enormous, spread across multiple files and hosts, and be cluttered with extraneous details. Additionally, examining logs from multiple executions independently, as opposed to side-by-side, can hide errors. These issues are exacerbated as the systems scale in size and complexity.

This work presents two tools, ShiVector and ShiViz, to help developers make better sense of their distributed traces. Figure 1 overviews the tools at a high-level. ShiVector augments logs with structured ordering information emphasizing communication between components. ShiViz leverages that information to generate concise visualizations of system behavior.

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
ACM 978-1-4503-2768-8/14/05
http://dx.doi.org/10.1145/2591062.2591134

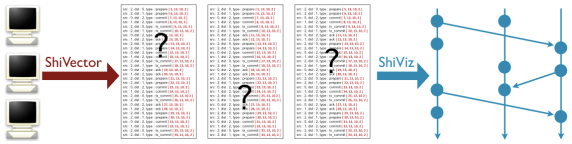

Figure 1: Logs generated by distributed systems are challenging to reason about. ShiVector augments such logs, as they are generated, with partial ordering information. ShiViz uses that information to visually summarize the system behavior.

One way to reason about the ordering of events in the distributed setting is with partial ordering [7], which defines *happens-before* relationships between pairs of events at different hosts. For any two log events e_1 and e_2, the partial ordering will indicate that either e_1 happens-before e_2, e_2 happens-before e_1, or e_1 and e_2 are concurrent. The happens-before relationship can provide useful insights into system behavior. Unfortunately, many systems do not maintain partial ordering information. We have developed **ShiVector**, a tool that automates the task of logging partial ordering information within the existing execution logs. ShiVector addresses two challenges. (1) Transparency: The developer gets access to the useful partial-order information without having to alter the system. (2) Correctness. The vector clock algorithm is nontrivial and it may be challenging, time-consuming, and error-prone to integrate this algorithm with the system logic.

A partial ordering by itself does not resolve the challenges associated with log analysis. The partial ordering provided by ShiVector adds structured meta-data to the log, but manually inspecting the log remains a laborious task. **ShiViz** uses the partial orderings provided by ShiVector to visualize a summary of a distributed system execution. These summaries provide developers with a high-level overview of a system's behavior by highlighting the ordering information that normally makes such logs difficult to reason about.

2. SHIVECTOR

ShiVector augments system execution logs with partial ordering information as the logs are generated (Figure 2). ShiVector's core

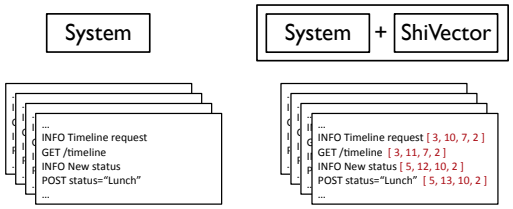

Figure 2: ShiVector does not change which events are logged by the system, but appends a vector timestamp to each event.

functionality is to manage a vector clock [5] for each host in the system and to add a vector clock timestamp to every logged event.

ShiVector, implemented in AspectJ, works on Java programs. It intercepts network and logging behavior without modifying the system's source code, although it does require recompiling the source with the AspectJ compiler.

3. SHIVIZ

ShiViz summarizes distributed system execution traces to give developers an overview of system behavior. The ShiViz visual summaries are concise visual models that emphasize the communication and ordering information that normally makes distributed systems difficult to reason about. ShiViz generates space-time diagrams, which relate events on different hosts based on the partial ordering information logged with ShiVector. Developers draw such diagrams to help understand the run-time behavior of a single execution of a system. Space-time diagrams can help developers tease out likely chains of causality, which is critical for debugging. ShiViz is deployed as a web service: `http://bestchai.bitbucket.org/shiviz/`

Figure 3 shows an example space-time diagram. Time flows from top to bottom. Vertical lines are process timelines, which represent threads of execution (e.g., processes). Events on a process timeline are events executed by that process. Edges connect events, representing the recorded happens-before relations. Paths in the diagram encode inferred happens-before relations: event e_1 happens-before event e_2 if and only if there is a monotonically downward path from e_1 to e_2. ShiViz is a web-based tool that developers can use to upload a log file and then explore the happens-before ordering, collapse process-local events to focus on communication, and view log lines that correspond to events in the diagram.

4. EVALUATION

We performed two case studies to evaluate the usefulness of ShiVector and ShiViz. First, we visualized the end-to-end integration tests of Voldemort[1], an enterprise distributed system. Second, we visualized a distributed query in SimpleDB, a parallel database developed as a course project.

ShiVector added partial ordering information to the Voldemort and SimpleDB execution logs without modifying any of the system's source code. We then used ShiViz to examine the interactions between threads in Voldemort. As an example observation, a common pattern emerged in which clients initiate a request, a server thread receives the request and forwards it to another server thread, which then replies to the original client. We observed this pattern 12 times — an easy feature to track in the ShiViz generated diagrams.

The ShiViz visualization quickly provided a useful summary of the system's behavior, though there are a number of features that we plan to add to the tool to improve its utility. For example, our current prototype of ShiViz is limited to visualizing a single system execution at a time. We plan to extend the tool to overlap multiple executions, highlighting differences between executions and potentially helping developers spot conditions under which deadlock or other undesirable conditions occur.

5. RELATED WORK

The Poet system [6], designed for debugging, proposes a toolchain for instrumenting a distributed system with vector timestamps and then visualizing the resulting traces as space-time diagrams. Though similar in design, our ShiVector and ShiViz tools are simpler to use and focus on improving developer comprehension of their

[1] `https://github.com/voldemort/voldemort`

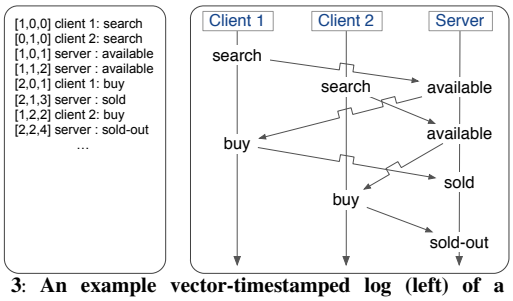

Figure 3: An example vector-timestamped log (left) of a system with two clients and one server, and a corresponding space-time diagram as would be produced by ShiViz (right).

systems. Our future work will evaluate our tools' effects on program comprehension by following the work of Cornelissen et al. [4].

A variety of tracing systems have been proposed to track requests in distributed systems [1, 2, 8, 9]. For example, vPath [9] is a VM monitor for understanding causality by tracking thread and network activity, and WAP5 [8] relinks a target application to custom system libraries to monitor network communication and reconstruct request paths using statistical inference. By contrast, ShiVector uses vector clocks to reconstruct the exact, partial ordering of events in the system without singling out requests. To understand a set of such traces CSight [3] can be used to infer a model of the distributed system that explains multiple observations. We plan to develop an alternative approach by adding a query interface to ShiViz to help developers query and explore multiple traces.

6. CONCLUSION

Logging is a common debugging technique, though the traces generated by distributed systems are often difficult to reason about manually. This paper briefly introduces two tools to help developers reason about distributed traces. **ShiVector** automatically augments distributed system execution logs with partial ordering information without modifying source code. **ShiViz** visualizes the augmented logs. Both tools are open-source:
`https://bitbucket.org/bestchai/shiviz`

7. REFERENCES

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. *SIGOPS OSR*, 37(5):74–89, 2003.

[2] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, 2004.

[3] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy. Inferring Models of Networked Systems from Logs of their Behavior with CSight. In *ICSE*, 2014.

[4] B. Cornelissen, A. Zaidman, and A. van Deursen. A controlled experiment for program comprehension through trace visualization. *TSE*, 37(3):341–355, 2011.

[5] C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *11th Australian Computer Science Conference*, pages 55–66, 1988.

[6] T. Kunz, J. P. Black, D. J. Taylor, and T. Basten. Poet: Target-System Independent Visualizations of Complex Distributed-Application Executions. *The Computer Journal*, 40(8):499–512, 1997.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, 1978.

[8] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. WAP5: Black-box performance debugging for wide-area systems. In *WWW*, 2006.

[9] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang. vPath: Precise discovery of request processing paths from black-box observations of thread and network activities. In *USENIX*, 2009.