X-ray: Root-cause Diagnosis of Performance Anomalies in Production Software

Mona Attariyan Michael Chow Jason Flinn
University of Michigan – Ann Arbor

Understanding and troubleshooting performance problems in complex software systems is notoriously challenging. This challenge is compounded for software in production for several reasons. To avoid slowing down production systems, analysis and troubleshooting must incur minimal overhead. Further, performance issues in production can be both rare and non-deterministic, making the issues hard to reproduce.

We argue that the most important reason why troubleshooting performance in production systems is challenging is that current tools only solve half the problem. Troubleshooting a performance anomaly is essentially the process of determining *why* certain events, such as high latency or resource usage, happened in a system. Unfortunately, most current analysis tools, such as profilers and logging [2, 5], only determine *what* events happened during a performance anomaly — they leave the more challenging question of why those events happened unanswered. Administrators and developers must manually infer the root cause of performance issue from the observed events based upon their expertise and knowledge of the software.

This poster describes X-ray, a tool that uses *performance summarization* to determine what events occurred during a performance anomaly and also why the anomaly occurred. Performance summarization first attributes performance costs such as latency and I/O utilization to fine-grained events (individual instructions and system calls). Then, it uses dynamic information flow analysis to associate each such event with a set of probable root causes such as configuration settings or specific data from input requests. The cost of each event is assigned to potential root causes weighted by the relative probability that the particular root cause led to the execution of that event. Finally, the per-cause costs for all events in the program execution are summed together. The end result is a list of root causes ordered by their performance costs.

X-ray can also compare the performance of two different requests. X-ray performs *differential performance analysis* to determine why two requests differed in performance. For instance, differential performance analysis can be used to understand why two requests to a Web server took different amounts of time to complete. Differential performance analysis identifies branches where the execution paths of the two requests diverged. It assigns a performance cost to each path taken from the branch, then uses dynamic information flow analysis to determine why the two requests diverged at that point. It attributes the difference in performance costs between the two paths to the identified root causes according to their relative likelihood. The costs of all such divergences are summed. The output of X-ray is a set of reasons of why the performance costs of two requests differ, along with a relative performance impact for each reason. Some prior research systems such as Spectroscope [3] also diagnose performance problems by comparing requests. Spectroscope, however, requires the requests to be the same; while X-ray is able to compare completely dissimilar requests and still identify the correct root cause.

Performance summarization is a high-overhead activity. In order to execute this analysis for production software, X-ray uses deterministic replay to offload the heavyweight analysis from the production system. A deterministic replay system records the execution of the system so that an identical execution can later be replayed on demand. While many prior software systems provide this functionality [4], our use of deterministic replay to troubleshoot performance issues raised several new challenges. X-ray must split its functionality among the recorded and replayed executions; for example, timestamps must be captured during recording because the heavyweight analysis substantially perturbs timing. Further, because of the split analysis, the fidelity of the replay must be strict enough to guarantee that the two executions are identical at the granularity observed by X-ray. However, because the replayed execution includes analysis code that the recorded execution does not, the fidelity of the replay must be loose enough to allow the replayed execution to diverge enough to run the analysis. X-ray achieves these goals through careful co-design of the deterministic replay and analysis systems.

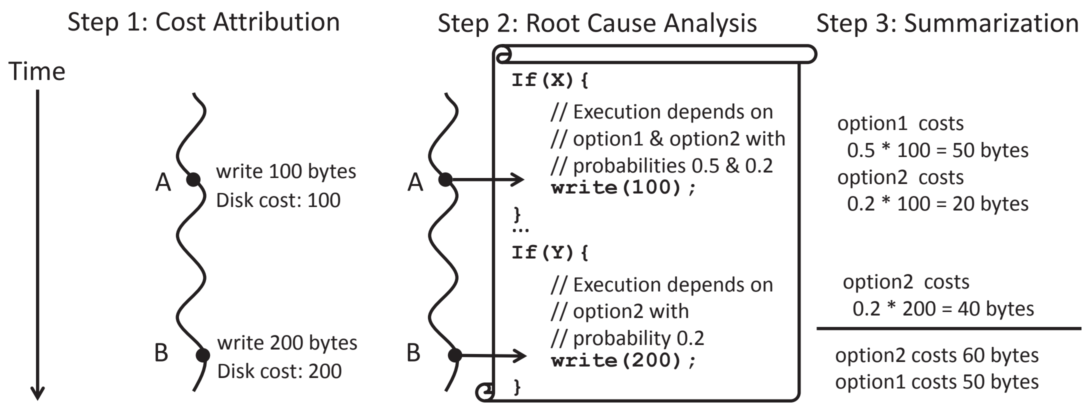

Figure 1. Overview of performance summarization

To troubleshoot with X-ray, the user first chooses which interval of execution to analyze. The user may select the entire execution, an interval of time, or a specific input request. X-ray produces a performance summary for the selected interval. Alternatively, a user may select two requests to compare, in which case X-ray does a differential performance summarization for the selected requests. The X-ray user next selects the set of performance statistics to summarize. Typically, we expect that a user will use basic performance analysis tools such as `top` and `iostat` to identify the bottleneck resource. X-ray provides a flexible framework for analyzing arbitrary statistics; our current implementation supports latency, CPU utilization, disk throughput and network bandwidth. Executions recorded by X-ray can be replayed multiple times. Therefore, X-ray users can perform many different analyses for the same recording.

Figure 1 shows an overview of how X-ray and performance summarization work. In the first step, X-ray attributes performance metrics to each event executed by one or more processes comprising a server application; the figure assumes that the X-ray user has specified disk bandwidth as a metric. Some metrics such as disk bandwidth are associated only with system calls, while others such as latency are attributed to both system calls and user-level instructions.

In the next step, X-ray uses taint tracking to derive a set of possible root causes for the execution of each event. Essentially, this step answers the question: "how likely is it that changing a configuration option or receiving a different input would have prevented this event from executing?" Taints are tracked using several algorithms developed in ConfAid [1]. To control the false positive rate, ConfAid uses numerical values to represent the strength of taints as they propagate. These numerical values follow two simple heuristics: data flow propagation is considered stronger than control flow propagation and control flow propagation of a recently executed branch is stronger than that of a less recently executed branch. These heuristics cause true positives to have larger taints than false positives.

In the last step, X-ray multiplies the performance metrics for each event by the per-cause taint values to derive the per-event performance cost for each root cause. X-ray sums these costs over all events that executed during the period selected by the user and outputs an ordered list of root causes.

So far, we have evaluated X-ray using three applications: the Apache Web server, the Postfix mail server and the PostgreSQL database. We have reproduced and analyzed 14 performance issues reported for these applications. In 12 of 14 cases, X-ray identifies a correct root cause as the largest contributor to the performance problem; in the remaining two cases, X-ray identifies a correct root cause as the third largest contributor. Our evaluation also shows that X-ray adds a performance overhead of only 1–7% on the production system.

References

[1] Attariyan, M., and Flinn, J. Automating configuration troubleshooting with dynamic information flow analysis. In *Proceedings of the 9th OSDI* (2010).

[2] Cantrill, B. M., Shapiro, M. W., and Leventhal, A. H. Dynamic instrumentation of production systems. In *USENIX Annual Technical Conference* (2004).

[3] Raja R. Sambasivan et al. Diagnosing performance changes by comparing request flows. In *Proceedings of the 8th NSDI* (2011).

[4] Xu, M., Malyugin, V., Sheldon, J., Venkitachalam, G., and Weissman, B. ReTrace: Collecting execution trace with virtual machine deterministic replay. In *Proceedings of MoBS* (2007).

[5] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordani, M. Detecting large-scale system problems by mining console logs. In *Proceedings of the 22nd SOSP* (2009).