2017 IEEE 37th International Conference on Distributed Computing Systems

milliScope: a Fine-Grained Monitoring Framework for Performance Debugging of n-Tier Web Services

Chien-An Lai *, Josh Kimball *, Tao Zhu *, Qingyang Wang #, Calton Pu *

* College of Computing, Georgia Institute of Technology
Atlanta, GA, USA,30332

Computer Science and Engineering, Louisiana State University
Baton Rouge, LA, USA, 70803

***Abstract*—Modern distributed systems are often considered to be black boxes that greatly limit the potential to understand behaviors at the level of detail necessary to diagnose some of the most important types of performance problems. Recently researchers have found abnormal response time delays, one to two orders of magnitude longer than the average response time, that exist in short periods and cause economic loss for service providers. These very short bottlenecks are hard to detect due to their short life spans and their variety of possible reasons. In this paper, we propose milliScope (mScope), the first millisecond-granularity software-based resource and event monitoring for distributed systems that achieves both performance, low overhead at high frequency, and high accuracy matched with other firmware monitoring tool. More specifically, milliScope is a fine-grained monitoring framework to collaborate multiple mScopeMonitors for event and resource monitoring to reconstruct the flow of each client request and profile execution performance in a distributed system. We utilize the resource mScopeMonitors for system resource monitoring, and we develop our own event mScopeMonitors to identify the execution boundary in a lightweight, precise and systematic methodology. The semantic and syntactic of these monitoring logs with arbitrary formats are enriched by our multistage data transformation tool, mScopeDataTransformer, which unifies the diverse monitoring logs into a dynamic data warehouse, mScopeDB, for advanced analysis. We conduct several illustrative scenarios in which milliScope successfully diagnoses the response time anomalies caused by very short bottlenecks using a representative web application benchmark (RUBBoS).**

I. INTRODUCTION

Previous researchers [1] [2] have found short-lived bottlenecks can introduce abnormal latency, i.e., system response times growing to 1 to 2 orders of magnitude greater than their average. According to an Amazon report [3], an increase of 100 milliseconds in system latency can lead to a 1% loss in sales. Isolating the root cause of these bottlenecks is challenging because of their fleeting nature and the large number of potential causes [4] [5].

Diagnosing very short bottlenecks in complex distributed systems necessitates researchers collecting measurements on many different system resources from potentially different monitors. For instance, individual, system-level monitors like SAR and IOstat can provide important system resource metrics at an individual node level [6]. Consequently, researchers need a framework to integrate and correlate these different monitors' measurements. Moreover, these measurements need to occur at very fine-grained timescales on the order of tens of milliseconds. The fact that no single, comprehensive utility exists speaks to the difficulty in diagnosing short-lived performance anomalies in large-scale systems.

In this paper, we present milliScope, the first millisecond granularity software-based resource and event monitor for distributed systems, which has both acceptable performance (low overhead at high measurement frequency) and high accuracy when compared to other firmware monitors, such as SysViz [7]. milliScope utilizes other, widely available monitoring tools, such as SAR, IOstat, Collectl, to monitor system resources at extremely fine-grained timescales. To capture each request's complete execution path and each node's complete execution profile in a complex distributed system, we develop our own lightweight event mScopeMonitors. These event monitors identify the execution boundaries of the requests. Our methodology is most similar to some other previous, excellent instrumentation techniques, such as Dapper [8], Magpie [9] and X-Trace [10]. Compared with these other approaches, our event mScopeMonitors impose negligible overhead by leveraging the native logging infrastructure accompanying each component server. Each request receives a unique identifier that accompanies the request as it

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.228

propagates across the system. As system components process requests, the corresponding unique identifiers are recorded at millisecond granularity in the components' logs, creating a composite of the components' execution boundaries.

Researchers need to be able to connect the critical points in a system's infrastructure to components' intact performance profiles to successfully debug performance anomalies. Constructing complete performance profiles requires a large number of measurements distributed over disparate monitoring logs to be merged and integrated. milliScope's fine-grained monitoring framework supports joining monitoring records generated by multiple mScopeMonitors. This integration enables researchers to analyze the distributed system performance across a wide variety of use cases. Concretely, milliScope contains its own data transformation tool, mScopeDataTransformer, which adopts a multi-stage parsing approach for enriching the semantics and syntax of ambiguous log messages. At the end of the pipeline, these semi-structured data are transformed into structured tuples and loaded into a dynamic data warehouse, mScopeDB. By encapsulating the diversity of monitoring tools through a uniform interface, milliScope is capable of correlating information across several system components at ideal granularity. Researchers are then able to use the collected and integrated information to more easily diagnose the root cause of performance anomalies.

We present two illustrative scenarios in which milliScope successfully diagnoses the response time anomalies caused by very short bottlenecks. These two scenarios look similar at first glance. They both exhibit requests with very long response times occurring over short time spans, but these long-running requests materialize due to different circumstances. In the first scenario, IO activities on the database server induce very long requests while the number of dirty page reaches a critical threshold on the web and application servers in the second situation. By integrating the tracing results from the resource mScopeMonitors and the event mScopeMonitors, milliScope provides the requisite monitoring data resolution to successfully diagnose response time anomalies caused by very short bottlenecks. Thus, the case studies demonstrate the benefits of milliScope: (1) it is able to zoom into the specific system components at fine-grained timescale granularity and (2) it can identify the root causes of very short bottlenecks, which provide opportunities for performance improvement.

II. MOTIVATION & RELATED WORK

Tracing tools play key roles in performance debugging and optimization of complex distributed systems like

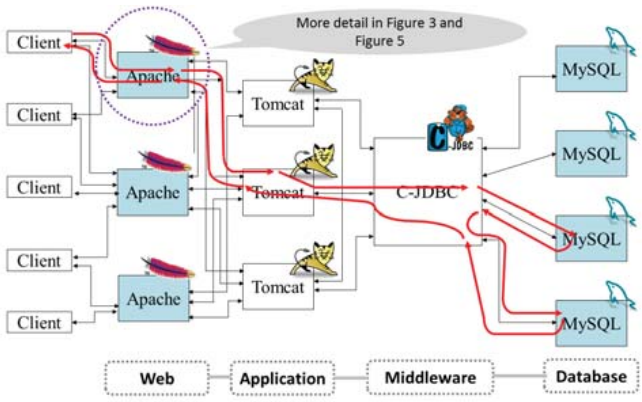


Fig. 1: An example of a four tier Web-App-Middleware-DB architecture with a possible causal path denoted as a thick line.

Figure 1. Profiling tools have traditionally focused on instruction-level operations within a single node context. For example, Paradyn and DTrace are two such tools [11] [12]. System-level tracing tools have been designed with either specific system models in mind or have operational constraints, which limit their utility. For example, tools in this latter group sometimes impose significant overhead on the instrumented system, causing broader system performance degradation [8]. Tools in the former category, which either rely on system operational models or on machine learning techniques to infer behavior from system throughput measurements, lack the necessary precision to diagnose millisecond-scale phenomena [13] [14]. Some tools like the one we use to validate own method, which provides the appropriate resolution for isolating these types of phenomena, lacks scale because of its rigid configuration requirements [7].

Recently, researchers have found that very short bottlenecks (VSBs, also called transient bottlenecks) can cause very long response time requests (VLRTs) , which are those that take one to two orders of magnitude longer to complete than average [1] [2]. The VSBs appear and disappear during a very short period of time, typically on the order of hundreds of milliseconds. Consequently, the VLRT requests also appear and disappear during these very short time periods. For example, Figure 2 shows the requests during this short interval have Point-In-Time response times that are more than twenty times the average. These VLRTs are often masked by the normal requests that only take a few milliseconds, particularly when the response time of requests is averaged over (typical) measurement periods of minutes. Current monitoring tools cannot capture and isolate VLRT requests, since they cannot provide fine-grained monitoring data

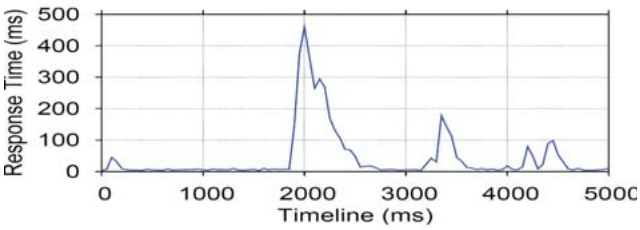


Fig. 2: The maximal Point-In-Time response time is more than twenty times larger than the average response time in the same period. If a monitoring tool samples at 1 second intervals, it would miss the response time fluctuations.

without degrading the system's overall performance.

Furthermore, diagnosing the root cause of VLRT requests is challenging due to the number of possible offending system resources. As previous works have shown, VLRT requests can occur for very different reasons. Potential root causes span different system levels, including CPU dynamic voltage and frequency scaling (DVFS) control at the architectural layer [4], Java garbage collection (GC) at the system software layer [2], virtual machine (VM) consolidation at the VM layer [5], and performance interference of memory thrashing [15]. As such, multiple resource monitors need to be deployed simultaneously to account for this resource and root cause diversity. For instance, we might simultaneously use SAR for CPU utilization, IOstat for IO activities, and Perf for memory bus usage.

Research has shown that a bottleneck cannot be detected using hardware utilization alone [16]. To study VSBs and the induced VLRT requests, we need an infrastructure capable of linking the monitoring data from resource monitoring tools to information about requests dependencies and causality at fine-grained time scales. By doing so, sampling methods can be avoided, since they can miss peaks like the one shown in Figure 2. In addition, the overhead imposed on the system-under-study by the method can be kept to a minimum. Previous end-to-end tracing implementations inserted metadata into requests to correlate individual system behaviors, and they relied on sampling to reduce overhead [17] [18] [19] [20] [21] [22]. A fine-grained framework on the other hand that captures the entire request execution map without needing to sample prevents assigning the wrong reason to a short-lived bottleneck. Black-box monitoring systems, which use statistical regression analyses to reconstruct causality without modifying traced systems, cannot meet these objectives [9] [23] [24] [25] [26] [14]. Although black-box methods incur low-overhead and do not require software modification, they are limited to specific workflows, since they rely on pre-built analytical models.

Fugitsu SysViz [7], which we use to validate the accuracy of our event mScopeMonitors in this paper, can reconstruct the entire trace for each transaction at sub-second levels, making very short bottleneck detection possible [27]. It uses special server hardware connected to network switches, which support passive network tracing, to collect its traces. Instead of hardware-based solutions for providing fine-grained monitoring functionality, milliScope provides the first software-based, millisecond-level resource and event monitoring solution for distributed systems. As we demonstrate later, it achieves comparable tracing accuracy (compared to SysViz) without comprising on scale or measurement frequency.

For a system phenomenon detection tool to be considered "complete," it should meet the following design objectives:

- The framework has to provide fine-grained monitoring data and distributed event correlation across a variety of native system logs.
- An interface that is able to easily reconstruct the causal path and profile the performance of each request.
- The overhead caused by the monitoring tool has to be negligible to prevent performance degradation.

III. MILLISCOPE

milliScope is a monitoring framework for n-tier applications–built specifically to document millisecond-level system phenomenon. It enables researchers to systematically reason about the relationships among individual component servers and corresponding resources on very small time scales. To achieve this objective, milliScope contains its own lightweight event mScopeMonitors to identify requests' execution boundaries, and it leverages an assortment of resource mScopeMonitors, such as Collectl, SAR and IOstat, to capture server components' resource utilization statistics. The disparate measurement data typically locked in various systems' log files can be imported into a dynamic data warehouse, mScopeDB, with mScopeDataTransformer, a multi-stage data transformation tool as shown in Figure 3. By connecting the instrumentation to the data transformation and integration pipeline, milliScope provides researchers with a unified interface for systematically identifying VLTRs, isolating corresponding resource bottlenecks and begin determining the underlying root causes for millisecond-level performance anomalies.

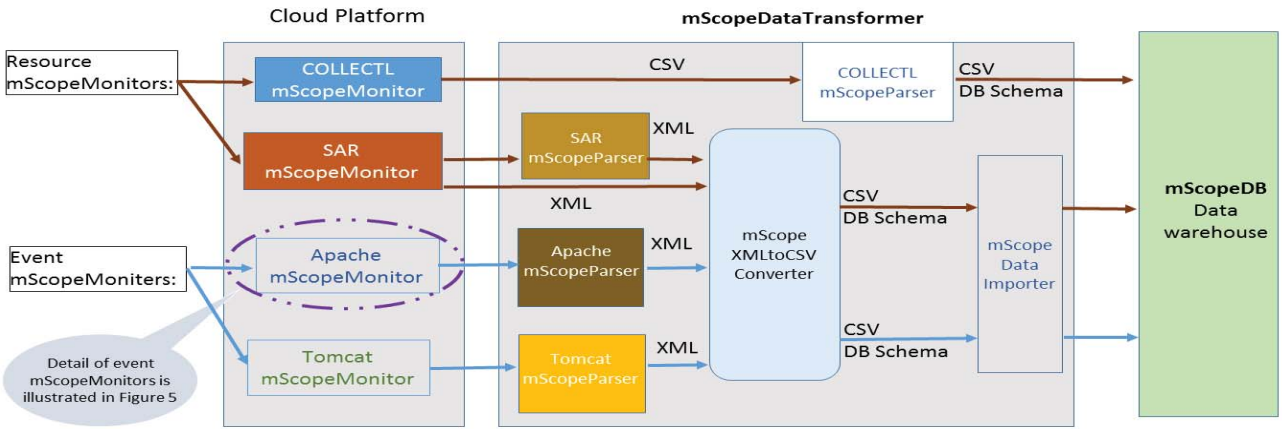


Fig. 3: The data transformation flow of milliScope. The event mScopeMonitors capture timestamps, as shown in Figure 5, in the component logs, while the resource mScopeMonitors record the system resource utilization. mScopeDataTranformer adopts a multi-stage approach to transform these unstructured data to structured tuples through different mScopeParsers, mScope XMLtoCSV Converter and mScope Data Importer respectively. And the monitoring data are loaded into a dynamic data warehouse, mScopeDB, for advanced analysis.

A. *Resource mScopeMonitors*

Applications produce a variety of resource consumption situations. To understand these usage and capability scenarios, milliScope uses several resource mScopeMonitors to monitor the utilization of targeted resources on specific system components. Currently, resource mScopeMonitors support a variety of resource monitoring tools such as SAR, IOstat and Collectl.

Each of the mScopeMonitors support different log formats. For example, SAR mScopeMonitor outputs monitoring data in pure text format by default, while Collectl mScopeMonitor is able to log its data in both plain text and csv file formats. The log format for these monitors is also affected by the resources a user chooses to monitor. For example, users might choose to CPU, memory, network, disk IO and process-level statistics or a subset of these. This choice can effect a given monitor's log format any number of ways, including the number of columns and rows and whether information like a header is repeated or not. milliScope handles this file structure and data format variability with mScopeDataTransformer.

B. *mScopeDataTransformer*

mScopeDataTransformer makes several passes over specified log files to transform the monitoring data into structured tuples, which can be loaded later into our dynamic data warehouse, mScopeDB. With each pass, additional semantics are added to the files to support a uniform downstream parsing activity. mScopeDataTransformer contains multiple customized parsers, converters and data importers to handle each of the different mScopeMonitors in the infrastructure. For example, SAR mScopeMonitors log files might be enriched over several passes with SAR-specific semantics, or files might be transformed directly through a one-pass customized parser like Collectl mScopeMonitors. We introduce the main stages of the data transformation in the following.

1) Parsing Declaration: Given the format and structure variability among the log files, the data extraction challenge is a non-trivial matter. We employ two techniques to help mitigate this complexity. We explicitly separate data extraction and log parsing from schema creation and data ingest. Our collection of parsers actually enrich their input log data with monitor-specific semantics, and their operation is governed by declarative-style instructions. mScopeDataTransformer maintains a mapping between input log files and their specific mScopeParser. Along with this parser-to-log file relationship, mScopeDataTransformer also records instructions for how the specified mScopeParser should inject semantics into its input logs. Currently, these parsers support adding semantics to files using either the sequence of lines in a file or specific string tokens (expressed as regular expressions). In short, during this stage, mScopeDataTransformer specifies which parser should be used for a given file and how the parser should inject meaning into said file.

2) Adding Semantics to Semi-structured Data: The next step in our process concerns mScopeParser execution. As shown in Figure 3, the Apache mScopeParser corresponds to Apache mScopeMonitor, and it begins by wrapping each line in the native log with the <log>

tag. This parser uses specific substrings as markers, so similar tags are added using the positions of other specific tokens. SAR mScopeMonitor has two paths presented in the figure. We originally built a customized SAR mScopeParser for SAR mScopeMonitor to infer its schema, because our two methods for providing instructions were insufficient. After we upgraded our version of SAR mScopeMonitor to a more recent version, we were able to directly output XML, obviating our custom approach.

3) From semi-structured Data to Structured Tuples: The final stage in our transformation pipeline turns semi-structured XML data into structured tuples. The mScope XMLtoCSV Converter is the component responsible for completing this final step. This component's XML interface separates the data annotation provided by specific mScopeParsers from the data warehouse schema creation. First, mScope XMLtoCSV Converter infers the table schema from the XML annotation. To materialize the data warehouse schema, we adopt a bottom-up approach. The number of columns for a table schema is based on applying the union operation to all of the metadata in the XML file. The datatype for a given column is determined using the best match principle, i.e., the narrowest data type that can store all of the values for the same XML tag is the one selected. Besides the inferred schema, the converter extracts the data from the XML file and outputs it to CSV files. The downstream data loader, entitled mScope Data Importer, uses these artifacts to create the tables and load the data tuples into the appropriate database table.

C. mScopeDB

mScopeDB is a dynamic data warehouse for persisting performance data generated by milliScope. Concretely, it uses four static tables to store data loading-metadata like environmental configuration and dynamically created tables to persist the data like CPU, Memory, Network and I/O originating from resource mScopeMonitors. The event mScopeMonitor data and the component boundary timestamps are also treated as another type of resource. As mentioned in the previous section, mScopeDataTransformer creates and populates these dynamic tables on-the-fly. Our dynamic data warehousing approach aims to hide some of the complexity associated with analyzing a large amount of performance data collected from a variety of sources. For instance, researchers might wonder if any disk activities happen during the period when Point-In-Time response time fluctuates heavily as in Figure 2. With mScopeDB, researchers are able to explore the disk utilization scenario across different component nodes, and observe in this case that the disk of the database node has reached full utilization during this short span, as shown in Figure 4.

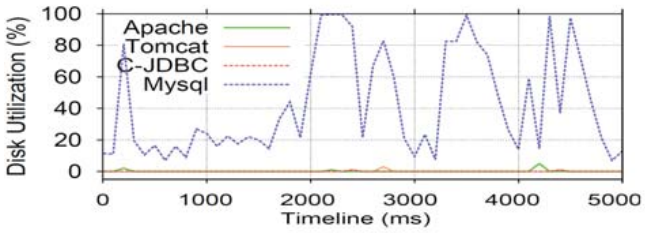


Fig. 4: Disk Utilization at different n-tier component nodes in the same period as shown in Figure 2. We observe that disk of Mysql has reached full utilization a couple of times during this period.

IV. EVENT mSCOPEMONITORS

A. Distributed Event Monitoring & Logging

In addition to the data transformation utility described in Section III, we have developed event mScopeMonitors–lightweight, scalable, and precise request flow tracing tools that can identify the execution boundary of each request. This comprehensive utility, which leverages existing logging infrastructure to minimize overhead, provides complete system component coverage. This enables these tools to reveal request dependencies and correlate events (generated by request activity) with resource mScopeMonitor data.

Each event mScopeMonitor modifies the component source code to collect request-specific execution information. Generally, it makes three types of code modifications using code specialization techniques: generating request-specific timestamps, adding logging to output timestamps and inserting unique identifiers into requests. The event mScopeMonitor has dual objectives: detect abnormal phenomena, like the one presented in the Point-in-Time response time graph in Figure 2, and provide sufficient information to support a detailed diagnosis of the problem. For example, to identify the server causing VLRT requests and contributing to queue amplification, we need to know the contribution of each server to the response time of each request.

B. Event of Interest

The first question is deciding how much information an event-logging monitoring tool needs to capture in order to re-create a request's set of related activities across a distributed system. Removing any unnecessary data also helps to reduce a monitoring tool's overhead–another goal of the event mScopeMonitors.

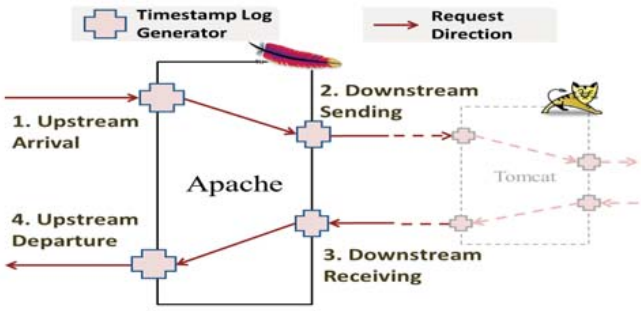


Fig. 5: Each event mScopeMonitor records four timestamps for each request on each component, which can be used to rebuild the causal relationship.

To accomplish this end, our approach records only four timestamps for each request on each component server that the request touches. These timestamps are as follows:

- Upstream Arrival timestamp: the timestamp when the request arrives at the component server from an upstream tier.
- Upstream Departure timestamp: the timestamp when the request is returned to an upstream server.
- Downstream Sending timestamp: the timestamp when the request leaves the component server for a downstream server.
- Downstream Receiving timestamp: the timestamp when the request is returned from a downstream server.

To identify a specific request's causally-related activities occurring across an n-tier system, Apache mScopeMonitor inserts a static, fixed-width request ID into the URL, and this request ID propagates to downstream tiers as a URL parameter or as part of a comment to a SQL query. By joining the tracing records containing the same request ID located in the event mScopeMonitor log files, milliScope is able to reconstruct the execution path explicitly, as Figure 5 shows. This enables milliScope to establish *happens-before* relationships among component servers in the system without making any assumptions about the interactions among servers. This data can also be used to calculate metrics useful for filtering potential bottlenecks. For example, once we calculate the instantaneous number of queued requests for each tier for the same period as Figure 2, we find the pushback phenomena occurs when the database tier's queue length increases concurrently with the other tiers', as shown in Figure 6.

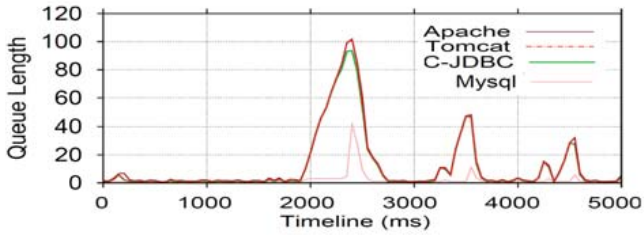


Fig. 6: Instantaneous # of queued requests for each tier for the same period as shown in Figure 2. Pushback is found, since the database queue length increases concurrently with the other tiers' increases.

C. Specialized Logging Facilities

Logging activities have been known to cause a dramatic reduction in performance by introducing significant overhead, since they involve a lot of CPU and IO operations [28]. Previous monitoring tools such as Dapper [8] and Zipkin [21] have used sampling to reduce their overhead. However, as we saw in Figure 2, VSBs (very short bottlenecks) probably only endure for tens or hundreds of milliseconds, and would not have been detectable with sampling intervals of seconds or minutes [27].

The event mScopeMonitors by design trace all request activities, so our tool needs to intelligently manage logging to limit its overhead. An intuitive and common approach for handling the IO associated with logging is to leverage the existing logging facility of a host, since it enables runtime logging without modifying the application binary. Concretely, the event mScopeMonitors modify the source code of software components to integrate the previously mentioned timestamps into existing log files. Using deliberate specification in the source code, the overhead can be reduced to 1% to 3% CPU utilization. We show the complete overhead comparison in Section VI-B, and we illustrate the detail of specialization using Apache as an example in Appendix A.

V. ILLUSTRATIVE SCENARIOS

In this section, we provide two illustrative scenarios for how milliScope accomplishes each of the following activities: collects data from the event mScopeMonitors and the resource mScopeMonitors, transforms the native logs into structured data through mScopeDataTransformer and loads it into mScopeDB. With milliScope, we are able to "scale the mountain" of data to look for the root cause of observed performance anomalies. In both scenarios, we discover the VSBs that cause the VLRT requests, and we show milliScope makes no assumptions

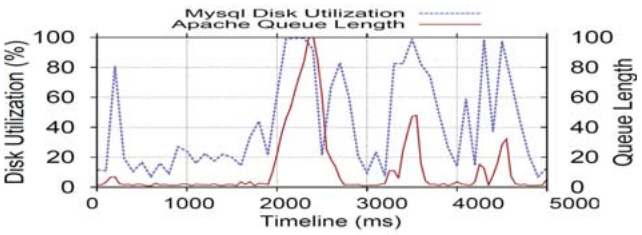


Fig. 7: Further investigation for Figure 2 via milliScope. Once the IO of the database tier is saturated because the database flushes logs from memory to disk, the requests at the Apache tier starts queueing. This figure shows disk IO is the very short bottleneck and makes the Point-In-Time response time increase dramatically during the very short bottleneck period.

about the origin of bottlenecks. This is exhibited by highlighting the different reasons for the bottlenecks in the scenarios, database I/O activities and memory thrashing respectively.

A. Database IO as the Very Short Bottleneck

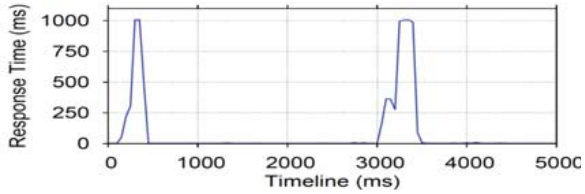
In our first case, we review the period in which the maximal Point-In-Time response time suddenly becomes twenty times larger than the average response time as shown in Figure 2. This period only exists for hundreds of milliseconds, and the Point-In-Time response time returns to normal quickly. Coarse-grained monitoring tools, such as periodically sampling at one second intervals, might overlook the peak and miss the opportunity for performance improvement.

To better understand the reason for such performance degradation, we calculate the instantaneous, concurrent requests in each tier using the monitoring data provided by the event mScopeMonitors. Other event monitoring tools cannot usually provide the correct number of concurrent requests, since they usually adopt sampling to reduce overhead. As depicted in Figure 6, obvious cross-tier pushback phenomena [27] occurs, evidenced by the database tier's and the other tiers' queue lengths increasing simultaneously. To investigate the reason why these elongated queues persist for hundreds of milliseconds, we apply Collectl mScopeMonitor to interrogate the resource utilization of each tier during this period. Since milliScope has transformed the native logs into structured tuples housed in our dynamic data warehouse, mScopeDB, we can easily associate monitoring data from several system components for the same period. As displayed in Figure 4, the disk utilization of the database tier varies dramatically, while the disk utilization of the other tiers remains consistently low. We conclude this case by showing the high correlation that exists between the disk utilization of the database and the Apache queue length found in Figure 7. This relationship provides strong evidence for the database IO causing the very short bottleneck. Previous research has shown short lifespan IO activity is triggered by the database flushing its logs from memory to disk in order to maintain consistency [29].

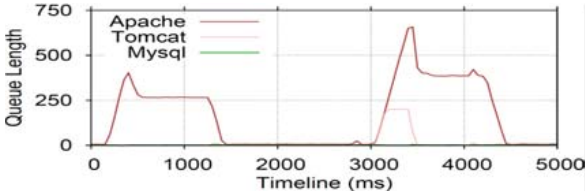
B. Memory Dirty-Page as the Very Short Bottleneck

A dramatic increase in Point-In-Time response time might be caused by different system components or system layers. With milliScope, researchers are able to utilize a variety of fine-grained resource mScopeMonitors and still integrate their data easily. In this section, we demonstrate another example of our system performance debugging system, milliScope, successfully detecting another performance anomaly. First, we observe the Point-In-Time response time reaches one thousand milliseconds twice while the average response time is less than twenty milliseconds during a five second interval as shown in Figure 8a. After identifying the execution boundary of each request with the event mScopeMonitors and calculating the request queue lengths for each tier in Figure 8b, we found two similar looking Point-In-Time response time peaks. These peaks however are actually caused by different system components in the n-tier system. Specifically, during the first peak, only Apache's queue length increases, while the queue lengths for both Apache and Tomcat increase during the second peak. In other words, cross-tier queue amplification is observed only at the second Point-In-Time response time peak.

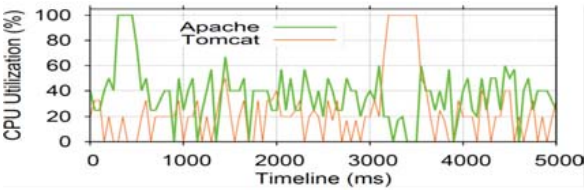
Checking the monitoring data from Collectl mScopeMonitor, we found the CPU utilization of Apache and Tomcat are saturated at the first and second peak respectively, as shown Figure 8c. However, the reason for CPU saturation differs from the previous case study (IO activities), since we don't observe high IO utilization in this period. milliScope is a fine-grained monitoring framework, which allows researchers to extend the monitoring scope easily. In this case, we utilize another subsystem in Collectl mScopeMonitor to understand the memory usage scenario. Once again, milliScope converts the native log of Collectl mScopeMonitor into structured tuples through mScopeDataTransformers multi-stage transformation prior to loading the data into the data warehouse. As shown in Figure 8d, the abrupt drop in dirty page cache size correlates with CPU saturation. This suggests Apache and Tomcat servers' separate dirty page recycling processes are the reason why Point-In-Time response times increase during these periods.



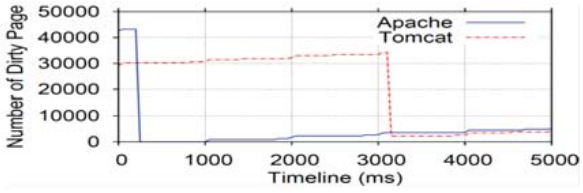
(a) Point-In-Time response time for a n-tier system reaches 1,000 millisecond twice while the average response time is less than twenty millisecond.



(b) Request queue length for each tier shows an interesting phenomena. During the first peak, only request queue length at Apache increases, but the request queue length at both Apache and Tomcat increase at the second peak.



(c) Checking the monitoring data through milliScope, we found CPU utilization of Apache and Tomcat are saturated at the first and the second peak respectively.



(d) Applying memory subsystem of Collectl mScopeMonitor, milliScope transforms the relative logs, such as number of dirty page, to structured tuples and loads them into mScopeDB. We demonstrate the root cause of CPU saturated is due to dirty page recycling.

Fig. 8: In a five second interval, we observe two peaks of Point-In-Time response time. Although they look similar at the first glance, they are actually caused by different components of an n-tier system.

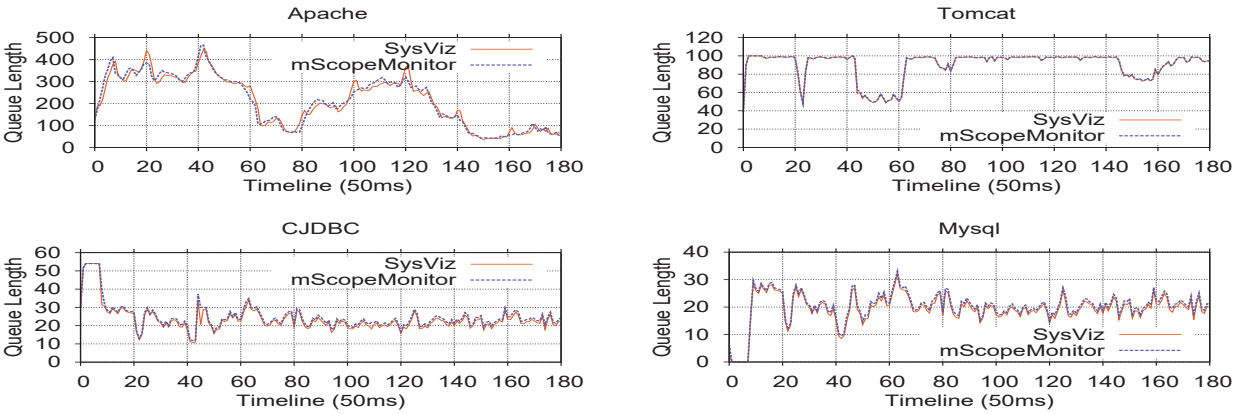
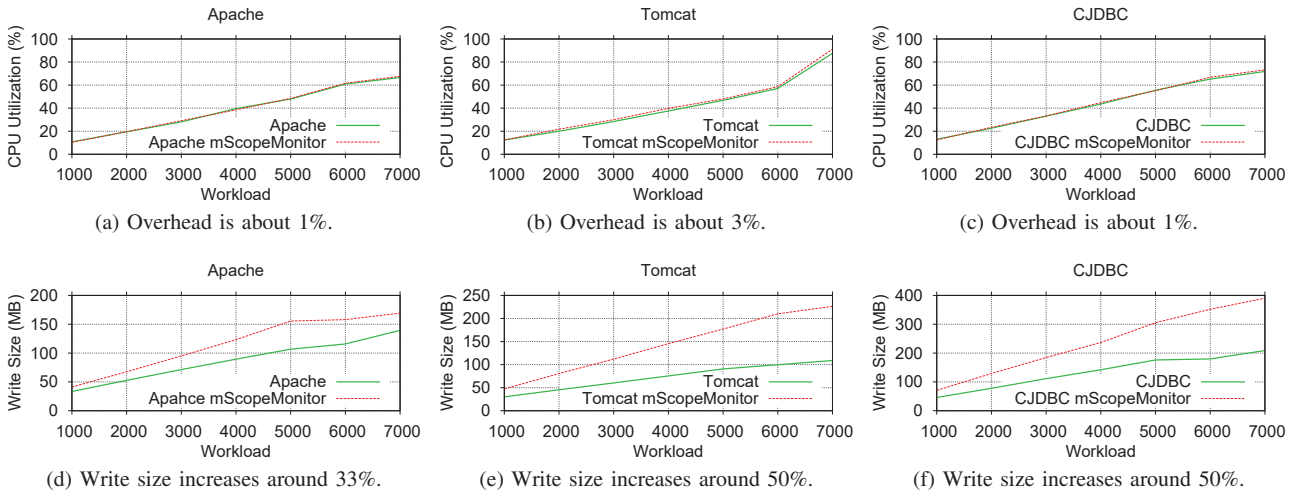


Fig. 9: Queue length comparison at workload 8000 between SysViz and the event mScopeMonitors among n-Tier systems, including Apache, Tomcat, CJDBC and MySQL. The event mScopeMonitors' results are very similar to SysViz's, which demonstrates the accuracy of the event mScopeMonitors.

VI. Evaluation

A. Accuracy Validation

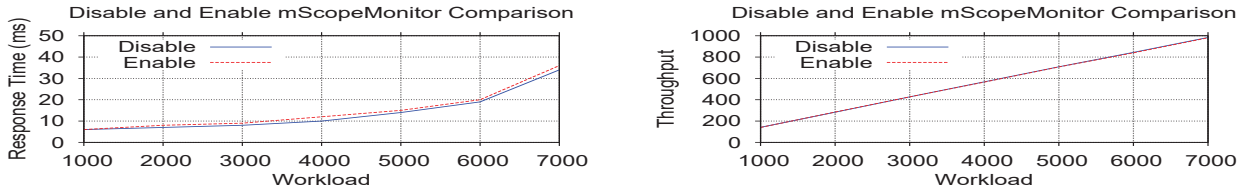
The event mScopeMonitors aim to provide just enough information to correlate the event information with the monitoring data generated by the resource mScopeMonitors on n-tier systems. These systems are typically organized as a pipeline of servers, starting at web servers, through application servers, middleware servers and ending with database servers organized in four tiers as shown in Figure 1. To validate the accuracy of each specific event mScopeMonitor, we compare its request queue length accounting for each system component with a commercial request tracing tool Fujitsu SysViz [7]. SysViz is able to reconstruct the entire trace of each transaction executed in a system based on interaction messages collected through network taps or network switches that support passive network tracing. In addition, we use RUBBoS, a standard n-tier benchmark [30], to simulate bulletin board applications such as Slashdot. The workload of RUBBoS consists of 24 different interactions such as "view story", and the value of the



(a) Overhead is about 1%. (b) Overhead is about 3%. (c) Overhead is about 1%.

(d) Write size increases around 33%. (e) Write size increases around 50%. (f) Write size increases around 50%.

Fig. 10: Compared to unmodified servers, the aggregated disk write size for event mScopeMonitors are up to two times, but they only increase 1% to 3% overhead.



(a) Response time increases about two milliseconds when mScopeMonitors are enable.

(b) Throughput shows slightly difference when mScopeMonitros are enable.

Fig. 11: Performance comparison between disable and enable mScopeMonitors using RUBBoS benchmark on a n-tier system, in which Apache, Tomcat, CJDBC and Mysql are running in one component node respectively.

workload represents the number of concurrent users. Each experimental trial is running for 7 minutes.

Due to space limitations, we only show the queue length, an important metric that can be derived from the request flow tracing data [27] for each tier at workload 8,000 as depicted in Figure 9. As these figures show, the event mScopeMonitors and SysViz determine very similar queue lengths for each tier regardless of the scenario. Consequently, this demonstrates milliScope's event mScopeMonitor's ability to trace requests accurately.

B. Overhead Comparison

We evaluate the impact of logging on system performance using three metrics: system throughput, system response time and IOWait as a component of overall CPU utilization. We investigate the impact of monitoring-related logging on system performance by comparing the performance of the RUBBoS [30] benchmark when the event mScopeMonitors are enabled on each of the component nodes of the underlying n-tier system. Whether the event mScopeMonitors are enabled or not, there is almost no difference in system throughput, as Figure 11 shows. Similarly, we compare the system response times for the same benchmark and underlying system. The instrumented system experiences two milliseconds more latency than its un-instrumented equivalent.

Figure 10 shows each node's respective IOWait via an aggregate CPU utilization metric, which includes the time the CPU spends in user mode, system mode and IOWait. Even though logging is not a computationally intensive task, an efficient logging method should not increase CPU IOWait. The graph depicts the magnitude of the IOWait penalty imposed by the event mScopeMonitors on the modified server components relative to their unmodified counterparts. We present these utilization measurements across a range of workloads to account for any decline in the percentage of idle time (and hence IOWait) as a consequence of larger workloads.

Apache mScopeMonitor and C-JDBC mScopeMonitor add about 1% overhead to their respective CPUs, which demonstrates that our monitors by integrating into the system's existing logging infrastructure impose

no additional IOWait penalty beyond what the logging infrastructure itself contributes. On the other hand, Tomcat mScopeMonitor adds about 3% to its CPU. The difference in overhead between Tomcat mScopeMonitor and the other mScopeMonitors is primarily due to an additional thread being created to record the timestamps associated with downstream server communication. Tomcat mScopeMonitor uses this extra thread to log variable-width data corresponding to the dynamic communication between Tomcat and the downstream servers. We also present in this graph the difference between the event mScopeMonitor-enabled components' aggregate disk write size and the corresponding unmodified components' disk write sizes for the same experiments and setup, as described in Section VI-A. Taking these figures together, we see a favorable tradeoff. Our event mScopeMonitors actually output twice as much data to disk, most of which is associated with the timestamps as shown in Figure 5, at the cost of increasing overhead 1% to 3% due primarily to increased IOWait. These evaluations demonstrate the event mScopeMonitor's ability to provide fine-grained monitoring data with only negligible overhead.

VII. CONCLUSION

In this paper, we present the first millisecond granularity, software-based resource and event monitor for distributed systems, milliScope. milliScope provides a fine-grained monitoring framework composed of different mScopeMonitors, mScopeDataTransformer and mScopeDB, which used together can provide a complete system performance profile. We present two illustrative scenarios in which the abnormalities look similar at first glance, e.g., response times increasing by one to two orders of magnitude over a short period, but we find these phenomena are due to different system operations: IO activities and dirty page recycling. We also validate the accuracy and lightweight characteristics of the event mScopeMonitors through several evaluations. With its good performance (low overhead at high frequency) and high accuracy, milliScope is an important contribution towards performance debugging of complex n-tier applications in cloud environments.

ACKNOWLEDGMENT

We thank the anonymous reviewers and our shepherd, Xiaosong Ma, for their feedback on improving this paper. This research has been partially funded by National Science Foundation by CISE's SAVI/RCN (1402266, 1550379), CNS (1421561, 1566443), CRISP (1541074), SaTC (1564097) programs, an REU supplement (1545173), Louisiana Board of Regents under grant LEQSF(2015-18)-RD-A-11, and gifts, grants, or contracts from Fujitsu, HP, Intel, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

APPENDIX

A. Specialized Apache Logging Facilities

We use Figure 5 to illustrate the sequence of events to log a request with Apache mScopeMonitor.

Since Apache is the first-tier among n-tier systems, it would insert a unique request ID into the URL and propagate it to downstream tiers. For example, the original request was:

http://rubbos/StoriesOfTheDay

Under Apache mScopeMonitor, the web server would generate a unique ID and attach it at the end of the url:

http://rubbos/StoriesOfTheDay?ID=XXX

The application server will retrieve the ID (by simple instrumentation) and send it to the corresponding SQL statement to retrieve related data, and the ID is included as part of a comment to the SQL statement:

SELECT id,title FROM stories /*ID=XXX*/

In terms of timestamps, the original Apache source code inherently records the Upstream Arrival and Upstream Departure timestamps for each request that it receives. These can be used to calculate the response time of each request; however, obtaining the intermediate Downstream Sending and Downstream Receiving timestamps for requests associated with Apache/Tomcat communication via *ModJK*, an Apache plugin for connecting to Tomcat, is non-trivial. First, we extend the response data structure *request_rec* in the standard header template include/httpd.h by adding variables for storing the Downstream Sending and Downstream Receiving timestamps as follows:

```
apr_time_t connector_stime;
```

Then, we modify mod_jk.c, the module responsible for communicating with Tomcat, by adding calls to the Apache Portable Runtime (APR) library to record the Downstream Sending timestamp and Downstream Receiving timestamp as follows:

```
r→connector_stime = apr_time_now();
```

Lastly, to output this added information (i.e., the Downstream Sending timestamp and Downstream Receiving timestamp variables added to *request_rec*) in the Apache log files, we modify modules/loggers/mod_log_config.c to log timestamps as follows:

```
apr_psprintf(" %" APR_TIME_T_FMT,
(r→connector_stime));
```

REFERENCES

[1] Q. Wang, Y. Kanemasa, M. Kawaba, and C. Pu, "When average is not average: large response time fluctuations in n-tier systems," in *9th International Conference on Autonomic Computing, ICAC'12, San Jose, CA, USA*, 2012.

[2] Q. Wang, Y. Kanemasa, J. Li, C. Lai, C. Cho, Y. Nomura, and C. Pu, "Lightning in the cloud: A study of transient bottlenecks on n-tier web application performance," in *2014 Conference on Timely Results in Operating Systems, TRIOS '14, Broomfield, CO, USA*, 2014.

[3] "Amazon found every 100ms of latency cost them 1% in sales." http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/.

[4] Q. Wang, Y. Kanemasa, J. Li, C. Lai, M. Matsubara, and C. Pu, "Impact of DVFS on n-tier application performance," in *Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems, TRIOS@SOSP 2013, Farmington, PA, USA, November 3, 2013*.

[5] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu, "An experimental study of rapidly alternating bottlenecks in n-tier applications," in *2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*, 2013.

[6] S. Godard, "Sysstat: System performance tools for the linux os, 2004."

[7] F. Laboratories, "Visualization in the Design and Opeartion of Efficient Data Centers." http://globalsp.ts.fujitsu.com/dmsp/Publications/public/E4_Schnelling_Visualization%20in%20the%20Design%20and%20Operation%20of%20Efficient%20Data%20Centers.pdf.

[8] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," Tech. Rep., 2010.

[9] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA*, 2004.

[10] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *4th Symposium on Networked Systems Design and Implementation (NSDI 2007), Cambridge, Massachusetts, USA, Proceedings.*, 2007.

[11] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The paradyn parallel performance measurement tool," *Computer*, vol. 28, no. 11, pp. 37–46, 1995.

[12] B. Cantrill, M. W. Shapiro, A. H. Leventhal *et al.*, "Dynamic instrumentation of production systems." in *USENIX Annual Technical Conference, General Track*, 2004, pp. 15–28.

[13] K. Shen, M. Zhong, and C. Li, "I/o system performance debugging using model-driven anomaly characterization." in *FAST*, vol. 5, 2005, pp. 23–23.

[14] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and M. Stumm, "lprof: A non-intrusive request flow profiler for distributed systems," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014.

[15] J. Park, Q. Wang, J. Li, C.-A. Lai, T. Zhu, and C. Pu, "Performance interference of memory thrashing in virtualized cloud environments: A study of consolidated n-tier application," in *2016 IEEE Ninth International Conference on Cloud Computing, San Francisco, CA, USA, June 27 - July 2, 2016*.

[16] Q. Wang, S. Malkowski, D. Jayasinghe, P. Xiong, C. Pu, Y. Kanemasa, M. Kawaba, and L. Harada, "The impact of soft resource allocation on n-tier application scalability," in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA*.

[17] P. Reynolds, C. E. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat, "Pip: Detecting the unexpected in distributed systems," in *3rd Symposium on Networked Systems Design and Implementation (NSDI 2006), San Jose, California, USA, Proceedings.*, 2006.

[18] A. Chanda, A. L. Cox, and W. Zwaenepoel, "Whodunit: transactional profiling for multi-tier applications," in *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal*, 2007.

[19] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: Tracking energy in networked embedded systems," in *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, San Diego, California, USA, Proceedings*, 2008.

[20] E. Thereska, B. Salmon, J. D. Strunk, M. Wachs, M. Abd-El-Malek, J. López, and G. R. Ganger, "Stardust: tracking activity in a distributed storage system," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/Performance 2006, Saint Malo, France*, 2006.

[21] "Twitter zipkin," http://http://twitter.github.io/zipkin/.

[22] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016.*, 2016.

[23] J. L. Hellerstein, M. M. Maccabee, W. N. M. III, and J. Turek, "ETE: A customizable approach to measuring end-to-end response times and their components in distributed systems," in *Proceedings of the 19th International Conference on Distributed Computing Systems, Austin, TX, USA*, 1999.

[24] S. Kavulya, S. Daniels, K. R. Joshi, M. A. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco: Statistical diagnosis of chronic problems in large distributed systems," in *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA*, 2012.

[25] B. 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 *2009 USENIX Annual Technical Conference, San Diego, CA, USA*, 2009.

[26] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA*, 2014.

[27] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu, "Detecting transient bottlenecks in n-tier applications through fine-grained analysis," in *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, Philadelphia, Pennsylvania, USA*, 2013.

[28] K. Veeraraghavan, D. Lee, B. Wester, J. Ouyang, P. M. Chen, J. Flinn, and S. Narayanasamy, "Doubleplay: Parallelizing sequential logging and replay," *ACM Trans. Comput. Syst.*, vol. 30, no. 1, p. 3, 2012.

[29] C. Lai, Q. Wang, J. Kimball, J. Li, J. Park, and C. Pu, "IO performance interference among consolidated n-tier applications: Sharing is better than isolation for disks," in *2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA*.

[30] "Rubbos: Rice university bulletin board system," http://jmob.ow2.org/rubbos.html.