MILLISCOPE: A FINE-GRAINED MONITORING FRAMEWORK FOR PERFORMANCE DEBUGGING OF N-TIER WEB SERVICES

A Thesis
Presented to
The Academic Faculty

by

Chien-An Lai

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
December 2017

MILLISCOPE: A FINE-GRAINED MONITORING FRAMEWORK FOR PERFORMANCE DEBUGGING OF N-TIER WEB SERVICES

DEDICATION

To my parents, Li-Shu and Chin-Hsing, you are my strong foundation

ACKNOWLEDGEMENTS

I would first like to thank my advisor, Dr. Calton Pu. He continuously and tirelessly guided me through the years and I deeply appreciate all his guidance, care, and sacrifice. He taught me the thrill of pursuing the unknown.

I would also like to thank my committee members, Dr. Ling Liu, Dr. Qingyang Wang, Dr. Shamkant B. Navathe and Dr. Ada Gavrilovska, for their support and feedback. I am grateful for the time they committed to helping me grow as a researcher.

I also thank the many collaborators I've enjoyed the opportunity to work with these years, both on this work and on other research projects: Dr. Jack Li, Dr. Junhee Park, Dr. Aibek Musaev, Tao Zhu and Josh Kimball. I am grateful for working with them through most of my graduate career. As we've shared an office together and collaborated on various research endeavors, we've also celebrated in each other's victories and helped each other through difficult challenges. We have accomplished much by working together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

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 delay, one to two order of magnitude longer time than the average response time, exists in short period and causes economical loss for service providers. These milliBottlenecks are hard to detect due to its short live span and its variety of possible reasons. In this thesis, 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 multi-stage 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 milliBottlenecks using a representative web application benchmark (RUBBoS). Besides, we validate the accuracy of our event mScopeMonitors and demonstrate availability and flexibility of milliScope through several evaluations.

CHAPTER I

INTRODUCTION

Tracing tools play key roles in performance debugging and optimization of complex distributed systems like Figure 1. However, existing tracing tools either are designed with specific system models in mind or have operational constraints, which limit their utility. For example, monitoring tools might have limitations such as: incurring high overhead that causes broader system performance degradation [67], lack scale due to particular configuration requirements [41], or lack precision because of an overreliance on machine learning techniques [86].

Recently researchers have found that milliBottlenecks (also called very short bottlenecks or transient bottlenecks) can cause very long response time (VLRT) requests, which have one to two orders of magnitude longer response times than average [76] [79]. The milliBottlenecks 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 longer than the average. These VLRT requests 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 without degrading overall system performance.

Furthermore, diagnosing the root cause of VLRT requests is challenging because of the variety of system resources that are potential candidates. As previous works have shown, VLRT requests can have very different causes which span different system levels, including

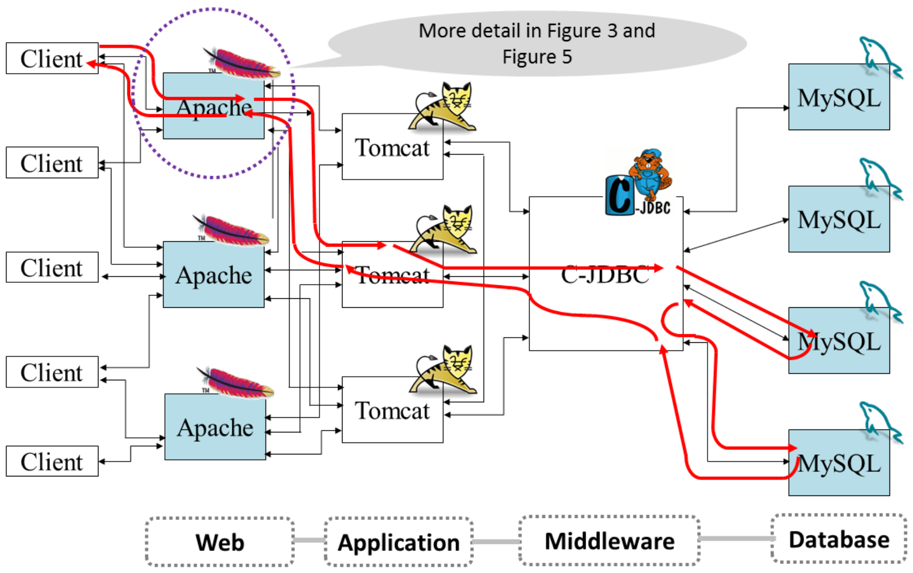


Figure 1: An example of a four tier Web-App-Middleware-DB architecture with a possible causal path denoted as a thick line.

CPU dynamic voltage and frequency scaling (DVFS) control at the architectural layer [80], Java garbage collection (GC) at the system software layer [79], virtual machine (VM) consolidation at the VM layer [78], and performance interference of memory thrashing [59]. As such, multiple resource monitoring tools have to be applied for measuring different system resources. For instance, we might simultaneously use SAR for CPU utilization, IOstat for IO activities, and Perf for memory bus usage.

Research has shown that system performance might quickly degrade due to the inappropriate allocation of soft resource, such as server thread pool size; hence, a bottleneck cannot be detected using hardware utilization alone [82]. To study milliBottlenecks and their associated 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. Such infrastructure can avoid the sampling methods since the sort of peaks shown in Figure 2 can then be missed. Moreover, the method has to impose minimal overhead on the system-under-study to prevent overall system performance degradation. Previous end-to-end tracing implementations inserted metadata to create correlation among individual system behaviors, and they relied on sampling to

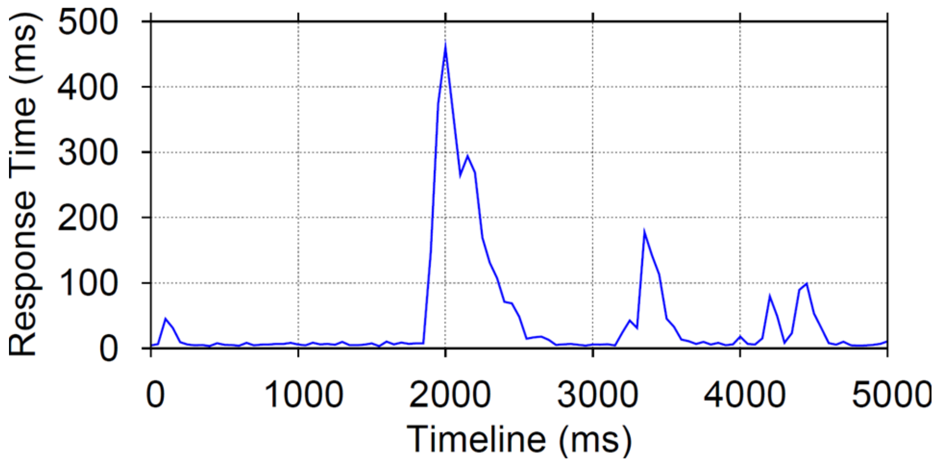


Figure 2: The maximal Point-In-Time response time is more than twenty times larger than the average response time in the same period. If a monitoring tool samples at 1 second intervals, it would miss the response time fluctuations.

reduce overhead [63] [16] [26] [71] [5] [51]. To avoid misunderstanding the reason for short-lived bottlenecks, a fine-grained framework that captures the entire request execution map without the need to sample is extremely important. Moreover, these objectives cannot be achieved by black-box implementations which use statistical regression analyses to reconstruct causality without modifying traced systems [12] [35] [39] [70] [20] [86]. Although this method incurs low-overhead and does not require software modification, it is limited to the specific workflows as it relies on a pre-built analytical model.

Fugitsu SysViz [41], which is also used to validate the accuracy of the event mScope-Monitors in this dissertation, can reconstruct the entire trace for each transaction at sub-second levels by connecting its server to network switches that support passive network tracing in order to make milliBottleneck detection possible [77]. Dislike the hardware-based solutions that can provide the fine-grained monitoring functionality, milliScope provides the first software-based, millisecond granularity resource and event monitoring solution for distributed systems. Our results show that milliScope not only achieves high degrees of accuracy at high frequencies without sampling, but also has better scalability than the hardware-based tracing tools.

Each of the mScopeMonitors might 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. These arbitrary log formats bring another issue that is how to provide a unified interface across various monitoring logs used for analysis. Researchers have proposed different methods for automatically parsing logs generated by servers and file systems [7] [84] [14]. Specifically, ExAlg constructs "templates" which are token strings that form equivalence classes; however, the templates are not used to create a schema or relations [10]. While this approach is similar to ours, the domain, semi-structured web-pages, is quite different in terms of both structure and related assumptions. In milliScope, we adopt a multi-stage data transformer which is not tied to a specific document structure, nor does it require meaningful user input in the form of regular expression creation.

1.1 Dissertation Statement and Contributions

The dissertation statement is formulated as follows:

Thesis Statement: *The impact of milliBottlenecks for n-tier application on modern compute cloud present the challenges of quality of service that can be detected, analyzed, and understood through an fine-grained monitoring software that achieves both performance, low overhead at high frequency, and high accuracy.*

In summary, our contributions is the fine-grained monitoring framework for performance debugging of n-tier web service, milliScope. To the best of our knowledge, milliScope is the first millisecond granularity software-based resource and event monitor for distributed systems. It provides a fine-grained monitoring framework composed of different mScopeMonitors, mScopeDataTransformer and mScopeDB, which used together can provide a complete system performance profile. More specifically, our contributions are as follows:

- We present a framework that provides fine-grained monitoring data and distributed event correlation across a variety of native logs. The framework contains resource monitoring tools, which help researchers check the utilization level of system resources, and the event monitoring tools that would be able to show the synchronicity of events and correlating variables. Inferring these properties from pre-existing logs or by making simplifying assumptions about the workflow model are not applicable, since these techniques would limit the effectiveness and usefulness of the monitoring framework.

  Besides, we provide an interface that is able to easily reconstruct the causal path and profile the execution performance for each request. Such an interface should relate the execution time the resource consumption of each node, thus enabling researchers to account for a significant fraction of the system's latency and to identify the component of a system bottleneck. Previous research has shown that a bottleneck cannot be detected using hardware utilization alone [82].

- We conduct several illustrative scenarios in which our fine-grained monitoring tool successfully diagnoses the response time anomalies caused by milliBottlenecks. With our framework, we are able to "scale the mountain" of data to look for the root cause of observed performance anomalies. In these scenarios, we discover the milliBottlenecks that cause the very long response time requests, but they occur for different reasons, database I/O activities and memory thrashing respectively.

- We demonstrate our event monitoring tools achieve low overhead and high accuracy simultaneously. As the first step, we validate the accuracy by comparing the request queue length for each system component between the event monitoring tools we implement and a commercial request tracing firmware, Fujitsu SysViz [41]. The results are very similar, which prove the accuracy of our monitoring tool.

  As the second step, we show the overhead comparison between the system components equipped with our monitoring tools and unmodified system components. The result shows our monitoring tools add about 1% to 3% cpu utilization only. The overhead caused by the monitoring tool has to be negligible to prevent performance degradation.

Table 1: The list of milliBottecks discovered by milliScope through multiple mScopeMonitors.

Memory Thrashing	Apache mScopeMonitor Tomcat mScopeMonitor Collectl mScopeMonitor
Database IO	Apache mScopeMonitor Tomcat mScopeMonitor C-JDBC mScopeMonitor Collectl mScopeMonitor
Dirty Pages in Memory	Apache mScopeMonitr Tomcat mScopeMonitor Collectl mScopeMonitor
Container Cross-Tier Overflow	Nginx mScopeMonitor Tomcat mScopeMonitor Collectl mScopeMonitor Lockstat mScopeMonitor

Meanwhile, the monitoring tool also has to record the execution history for each request. Reducing the overhead by instituting sampling is not an option, because the milliBottleneck could appear and disappear between sampling intervals due to its extremely short lifespan.

We have successfully discovered several milliBottlenecks as listed in Table 1 though milliScope.

Portions of this research were previously published and presented. milliScope, which is the first millisecond granularity software-based resource and event monitor for distributed systems, in the 37th International Conference on Distributed Computing Systems (ICDCS'17) [42]. Besides, milliScope has been applied on performance debugging of complex n-tier applications in cloud environments, and it has been successfully made contribution towards fine-grained system researches [60] [81] [88].

1.2 Outline

This dissertation is organized as follows. In Chapter 2, we present "milliScope" which is the first millisecond-granularity software-based resource and event monitoring for distributed systems. We discuss the components of milliScope, including the resource mScopeMonitors, the event mScopeMonitors, the data transformation platform mScopeDataTransfromer, and our dynamic data warehouse mScopeDB. Chapter 3 conducts several illustrative scenarios in which milliScope successfully diagnoses the response time anomalies caused by milli-Bottlenecks. Chpater 4 demonstrates more evaluations to validate the accuracy and the low oveerhead of our mScopeMonitors. In Chapter 5, we provide background on end-to-end tracing among the components of a distributed system. We conclude with some final remarks in Chapter 6.

CHAPTER II

MILLISCOPE

milliScope [42] is a monitoring framework for n-tier applications which is built to document system phenomenon at millisecond granularity. It enables researchers to systematically reason about the relationships among individual component servers and corresponding resources. Section 2.1 introduces the past and the challenges of milliBottleneck research and Section 2.2 shows the data flow of milliScope. We present both resource mScopeMonitors (Section 2.3) and event mScopeMonitors (Section 2.4). We conclude that milliScope provides a unified interface for researchers to perform advanced analysis on these data more easily in Section 2.6.

2.1 Introduction

Previous researchers [76] [79] 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 [1], 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 [80] [78].

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 [32]. 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 chapter, we present milliScope, the first millisecond granularity software-based

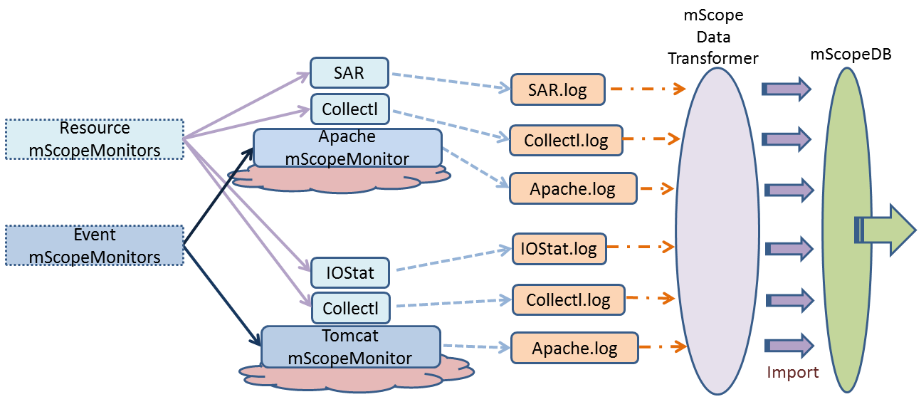


Figure 3: The data transformation flow of milliScope. The event mScopeMonitors capture timestamps, as shown in Figure 4, in the component logs, while the resource mScopeMonitors record the system resource utilization. mScopeDataTranformer converts these unstructured data to structured tuples and loads them into a dynamic data warehouse, mScopeDB, for advanced analysis.

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 [41]. 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 [67], Magpie [12] and X-Trace [28]. 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 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.

The monitoring facility needs to capture each request while minimizing the amount of overhead it imposes on the system-under-study to prevent overall system performance degradation. A lightweight, fine-grained monitoring tool needs to achieve the following objectives in order for it be considered "complete:"

- Reconstruct the causal path for each request by reviewing the execution time spent on each nodeenabling an administrator to account for a significant fraction of the systems latency and identify the location of a system bottleneck. Previous Research has shown that a bottleneck cannot be detected using hardware utilization alone.

- The overhead caused by the monitoring tool has to be negligible to prevent performance degradation. On the other hand, the monitoring tool has to record the execution history for each request. Reducing the overhead by instituting sampling is not an option, because the very short bottleneck could appear and disappear between sampling intervals due to its extremely short lifespan.

- The monitoring tool can be applied in a general workflow. Knowledge of the specific middleware or applications can be acquired in order to do instrumentation, but inferring causality by using the synchronicity of events, by correlating variables from pre-existing logs or by making simplifying assumptions about the workflow model are not applicable, since these techniques would limit the effectiveness and usefulness of the monitoring tool.

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.

2.2 milliScope Data Flow

The main data flow of milliScope is presented in Figure 3. The resource mScopeMonitors record the system resource utilization, while the event mScopeMonitors capture timestamps as shown in Figure 4. While the details of the resource mScopmonitors and the event mScopeMonitors would be discussed later in Section 2.3 and Section 2.4 respectively, we would like to describe the data transformation in milliScope mainly here.

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.

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.

Analyzing the performance and scalability measurements of n-tier applications is a tedious process. Researchers often do not know prior to doing an analysis which resource is responsible for creating a particular bottleneck. For example, an issue might be caused by CPU at first glance, but after more detailed data analysis, researchers might find most of the CPU utilization is attributed to waiting for I/O activities. This small example highlights some of the performance diagnosis challenges. First, it demonstrates the need to collect data from a variety of data sources. It also demonstrates the need to integrate this data across space and time to correctly isolate and diagnose performance anomalies. Our flexible, dynamic approach to data warehouse schema creation addresses both of these requirements. The dynamically created tables provide the flexibility for storing the variety of data coming from many different mScopeMonitors, such as SAR mScopeMonitor or Apache mScopeMonitor. Secondly, the necessary relationships for integrating the data can be built as the tables are constructed, since the data warehouse schema is built from the bottom-up one table at a time. With mScopeDB, users have the necessary tools to identify abnormal patterns in system performance and debug accordingly.

2.3 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.

Different mScopeMonitors use different log formats. For example, SAR mScopeMonitor reports monitoring data in pure text format by default, while Collectl mScopeMonitor is able to log monitoring data in both plain text and csv file formats. The number of possible monitoring variables that users can arbitrarily add to the variability of the log file format and structure. Users might decide to monitor CPU-related variables, or they might want to only monitor overall CPU and IO utilization. The variety of log format and the volume of data increase the difficulty of analysis. milliScope manages the sundry file structures and data formats through mScopeDataTransformer.

2.4 Event mScopeMonitors

2.4.1 Distributed Event Monitoring & Logging

In addition to the data transformation utility described in Section 2.2, 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

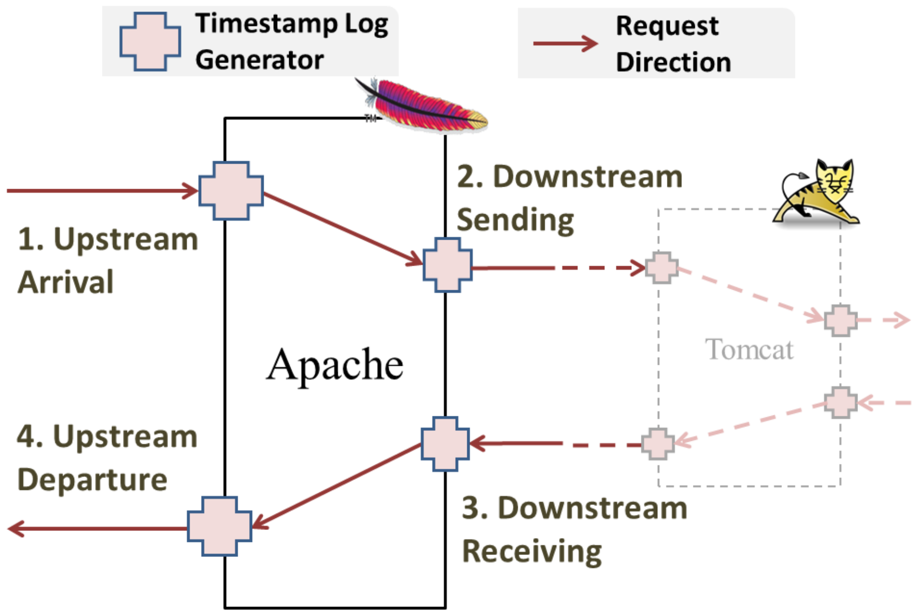


Figure 4: Each event mScopeMonitor records four timestamps for each request on each component, which can be used to rebuild the causal relationship.

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.

2.4.2 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.

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 4 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 7.

2.4.3 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 [72]. Previous monitoring tools such as Dapper [67] and Zipkin [5] 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 [77].

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 illustrate the detail of specialization using Apache as an example in Section 2.4.4.

2.4.4 Specialized Apache Logging Facilities

We use Figure 4 to illustrate the sequence of events to log a request with Apache mScope-Monitor.

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));
```

2.4.5 Specialized Nginx Logging Facilities

The code specialization of Nginx follows the same methodology as shown in Figure 4. Nginx as well as 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. The detail of request ID retrieve and insertion has been illustrated in Section 2.4.4.

Again, In terms of timestamps, like Apache, the original Nginx 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 Nginx/Tomcat communication is non-trivial. First, we extend the response data structure *ngx_http_upstream_state_t* in the standard header template http/ngx_http_upstream.h by adding variables for storing the Downstream Sending and Downstream Receiving timestamps as follows:

```
time_t TCST_sec;

ngx_uint_t TCST_msec;
```

```
time_t TCET_sec;

ngx_uint_t TCET_msec;
```

The time_t type variables would be responsible for up to second level granularity, while the ngx_uint_t type variable would store the millisecond part for the corresponding timestamps. Then, we modify http/ngx_http_upstream.c. When nginx is ready to forward the request to the component nodes at the downstream tier, it records the Downstream Sending timestamp from ngx_time_t data structure as follows:

```
u→state→TCST_sec = tp→sec;

u→state→TCST_msec = tp→msec;
```

, while Nginx receives the response from downstream tier node, it records the Downstream Receiving timestamp as follows:

```
u→state→TCET_sec = tp→sec;

u→state→TCET_msec = tp→msec;
```

These timestamps have to be encompassed into a string that would be logged into log file eventually. To achieve that, we add one line of source code in function ngx_http_upstream_response_time_var as follows:

```
p = ngx_sprintf(p, ”TCST(ms)=%03M TCET(ms)=%T%03M”,

state[i].TCST_sec, state[i].TCST_msec,

state[i].TCET_sec, state[i].TCET_msec);
```

Lastly, to output this added information (i.e., the Downstream Sending timestamp and Downstream Receiving timestamp variables in the string variable p) in the Nginx log files, we modify the function ngx_http_log_request_time in the file http/modules/ngx_http_log_module.c to log timestamps as follows:

```
return ngx_sprintf(buf, ”ST(ms)=%T%03M ET(ms)=%T%03M”,

r→start_sec, r→start_msec, tp→sec, tp→msec);
```

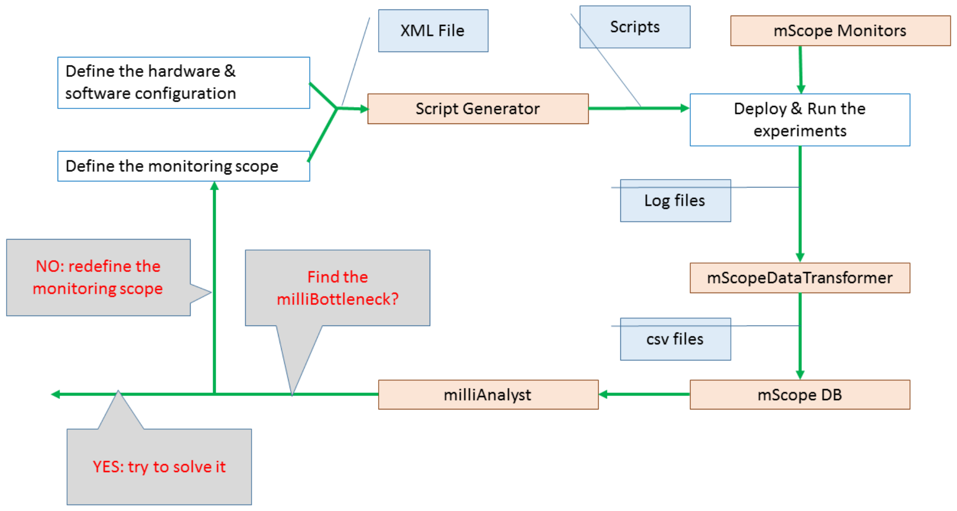

XML File
Scripts
mScope Monitors
Define the hardware & software configuration
Script Generator
Deploy & Run the experiments
Define the monitoring scope
Log files
NO: redefine the monitoring scope
Find the milliBottleneck?
mScopeDataTransformer
csv files
milliAnalyst
mScope DB
YES: try to solve it

Figure 5: The work flow of milliBottleneck discovery. Users define the configuration file at first, and the script generator generates scripts which set up the experiment environment and deploy milliScope as well as other softwares. mScopeDataTransformer converts these unstructured data to structured tuples in mScopeDB as described in Figure 3 for advanced analysis.

2.5 Open-source milliScope

We have released the source code of the milliScope as well as related scripts of the experiments on our website [3], and we aim to help researchers detect the milliBottlenecks on more scenarios easily. The performance unpredictability associated milliBottlenecks has impeded the migration of the applications from in house infrastructures to public clouds. By turning milliScope as an open-source project, more researchers could utilize our fine-grained monitoring framework on other scenarios to validate if there are more milliBottlenecks existing. Besides, through turning milliScope as an open source project, researchers are able to contribute more mScopeMonitor components, port them to other platforms, and achieve the growth of the milliScope ecosystem. Figure 5 describes the scope of source code releasing and the work flow of milliBottleneck discovery. The users first define the configuration files, and then the script generator would generate the corresponding scripts to set up the experimental environment and deploy mScopeMonitors as well as other softwares. Once the experiment is finished, mScopeDataTransformer collects the native logs and converts these unstructured data to structured tuples in mScopeDB, while milliAnalyst would investigate the reason of the performance variation if it exists. If the root cause of milliBottleneck was out of the scope of monitoring, the users would refine the configuration files and rerun the experiments. Currently, we are on the path to release the source code of these components, including script generator, mScopeMonitors, mScopeDataTransformer and milliAnalyst. Researchers would be able to quantify the impact of the milliBottlenecks and improve the Quality of Service (Qos) of the could service easily in the future.

2.6 Conclusion

This chapter presents the first part of my core thesis research that is “milliScope”, the first millisecond-granularity software-based resource and event monitoring for distributed systems. First, Section 2.2 briefly discusses the data transformation platform, mScop-DataTransformer, and our dynamic data warehouse, mScopeDB. After that, Section 2.3 describes the resource mScopeMonitors, while Section 2.4 discloses the details of our own event mScopeMonitors. milliScope provides a fine-grained monitoring framework composed of different mScopeMonitors, mScopeDataTransformer and mScopeDB, which used together can provide a complete system performance profile.

CHAPTER III

MILLIBOTTLENECK

This chapter presents the second study of my core thesis research that is the two illustrative scenarios of milliBottleneck detection when we apply milliScope to monitor the system utilization and related events and diagnose the root cause of the milliBottleneck. In Section 3.1, we introduce the background of milliBottlenecks, while we describe our experiment setup in Section 3.2. Section 3.3 discovers that database IO activities results in the drastic increase in Point-In-Time response time which grows from 20 ms to just less than 500 ms in hundreds of millisecond interval. In Section 3.4, we demonstrate that memory thrashing should be responsible for the milliBottlenecks occurring at two different tiers of a n-tier application in five second period. For both cases, milliScope makes no assumption about the origin of milliBottlencks but discovers the system component that cause the VLRT requests successfully. Finally, Section 3.6 provides an overview of related work in this area, and Section 3.7 concludes this chapter.

3.1 Introduction

milliBottlenecks emerge in web-facing applicatons and cause long-tail latency problem [75] when a majority of normal queries responding within milliseconds appear with a non-trivial number of queries with very long response time (VLRT), on the order of seconds. Although there have been several studies on various aspects of milliBottlenecks, practitioners continue to report real-world problems recently [22] [38] [46] [85]. Moreover, due to the management concerns with milliBottlenecks and long-tail latency, data centers persist low dutilization levels to prevent from the very long response time [45] [49].

The technical challenges in milliBottleneck research arise from the lacking of fine-grained monitoring tools and from the variety causes of milliBottlenecks, which can be divided into three categories. First, the uneven resource is required for n-tier applications with apparently uniform workload [37]. For example, web search of popular terms can return many more results than normal terms. Second, the resource contention in single nodes is triggered by busty workload, such as interference by “noisy neighbors” [58] [73] [83]. Third, the resource contention is amplified by dependencies among distributed nodes, a phenomenon known as “Cross-Tier Queue Overflow”. Although the execution of one request by itself would only take milliseconds, it might be observed as the very long response time request (VLRT) under moderate average resource utilization (e,g,. 50%) of all participating nodes. Therefore, milliBottlenecks may arise independently even in distributed systems where the average system utilization is far from saturation. Due to the dependencies and interactions among components, the performance of distributed systems is far more complex than a single server’s behavior [55].

To study the milliBottlencks and the induced very long response time (VLRT) requests, we need an infrastructure capable of providing fine-grained monitoring data and linking these data to information about requests dependencies and causality so that the researchers are able to do advanced analysis easily. Here, we have to point out that sampling methods are incapable of such fine-grained monitoring task since they can miss peaks like the one shown in Figure 2. To make milliBottleneck detection possible, Fugitsu SysViz uses special server hardware connected to network witches to trace the packets of the request [41]. Instead of hardware-based solution, we propose milliScope as the first software-based millisecond-level resource and event monitoring solution for distributed system.

As previous works have shown, VLRT requests can occur for very different reasons, including CPU dynamic voltage and frequency scaling (DVFS) control at the architectural layer [80], Java garbage collection (GC) at the system software layer [79], virtual machine (VM) consolidation at the VM layer [78], and performance interference of memory thrashing [59]. In this chapter, we provide two illustrative scenarios in which milliScope: collects the 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 that the milliBottlenecks cause the VLRT requests, but they occur for different reasons, database I/O

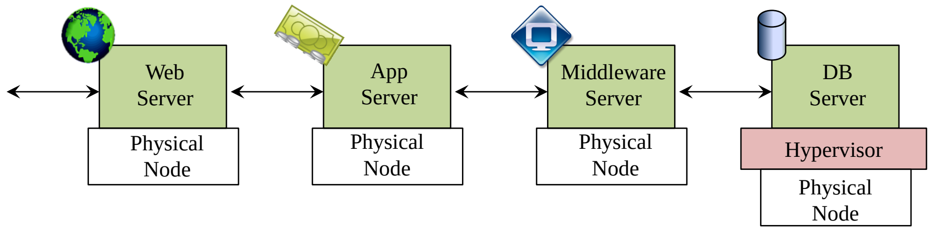


Figure 6: Dedicated deployment of a 4-tier application system with four software servers (i.e., web, application, middleware, and database) and four physical hardware nodes

activities and memory thrashing respectively.

3.2 Experimental Setup

While consolidation in practice may be applied to any type of application, the focus of this chapter is n-tier applications with LAMP (Linux, Apache, MySQL, and PHP) implementations. Typically, n-tier applications are organized as a pipeline of servers, starting at web servers (e.g., Apache), through application servers (e.g., Tomcat), and ending with database servers (e.g., MySQL) organized in three tiers or four where an additional layer contains an application, or middleware, for clustering (e.g., C-JDBC). This organization, commonly referred to as n-tier architecture (e.g., 4-tier in Figure 6), serves many important web-facing applications such as e-commerce, customer relationship management, and logistics.

In our experiment, we deploy the popular n-tier application benchmark system RUBBoS [4], based on bulletin board applications such as Slashdot. RUBBoS has been widely used in numerous research efforts due to its real production system significance. The workload includes 24 different interactions such as “view story” and “register user”. The benchmark includes two kinds of workload modes: browse-only and read/write mixes. In this chapter, we focus entirely on the read/write workloads, and specifically, the ratio of write request to all is 10%. Our default experiment trial consists of a three-minute ramp-up, a three-minute runtime, and a thirty-second rampdown. The hardware and software specifications are listed in Table 2.

[0]In this chapter, we refer to server in the context of computer programs serving client requests. Hardware is referred to as *physical computing node* or *node* for short.

Table 2: Summary of experimental setup (i.e., hardware, operating system, software).

CPU	2 Quad Q9650 3GHz * 4CPU
Memory	16GB
HDD	SATA, 7,200RPM, 500GB
Network I/F	1Gbps
Web Server	HTTPD-2.2.22
App Server	Apache-Tomcat-5.5.17
Connector	Tomcat-Connectors-1.2.32-src
Cluster middleware	C-JDBC 2.0.2
DB Server	MySQL-5.5.19-Linux2.6-i686
Java	JDK1.6.0_27
Monitoring Tools	Collectl
Operating System	RHEL Server 6.3 64-bit
OS Kernel	2.6.32-279.22.1.e16.x86_64

3.3 Database IO as the milliBottleneck

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. Specifically, we use window with 50 millisecond window size, and the requests are attributed to designated window depending on the starting timestamp. For example, if there are three windows representing the time interval from 0 to 50, from 51 to 100 and from 101 150 respectively, and if there is a request staring at 40 millisecond timestamp and ending at 120 millisecond timestamp, this request will increase the number of instantaneous queue length for the window number one (e.g. from 0 to 50) since we use starting timestamp to decide which window the request belongs to.

Other event monitoring tools cannot usually provide the correct number of concurrent

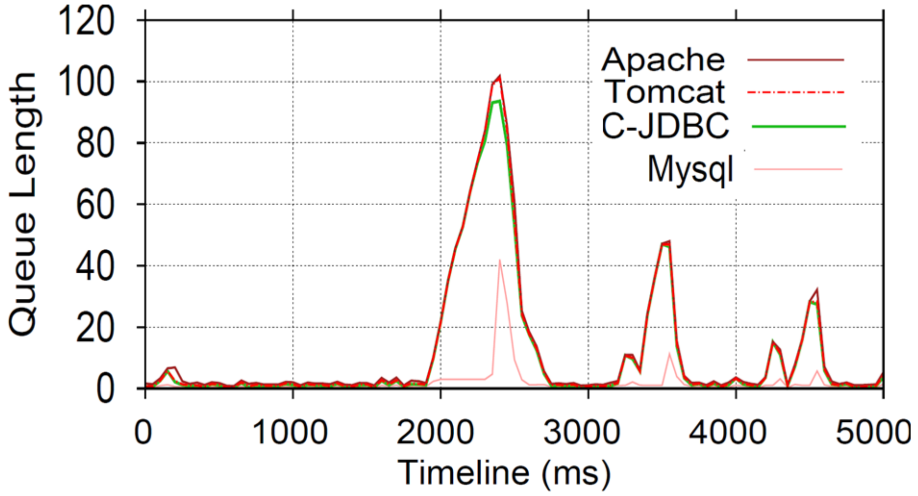


Figure 7: Instantaneous # of queued requests for each tier for the same period as shown in Figure 2. Pushback is found, since the database queue length increases concurrently with the other tiers' increases.

requests, since they usually adopt sampling to reduce overhead. As depicted in Figure 7, obvious cross-tier pushback phenomena [77] happens, evidenced by the concurrent increasing queue lengths of the database tier and the other tiers. To investigate the reason why the queue length persists 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 across several system components during the same period. As displayed in Figure 8, 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 9. This relationship provides strong evidence for the database IO causing the milliBottleneck. 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 [43].

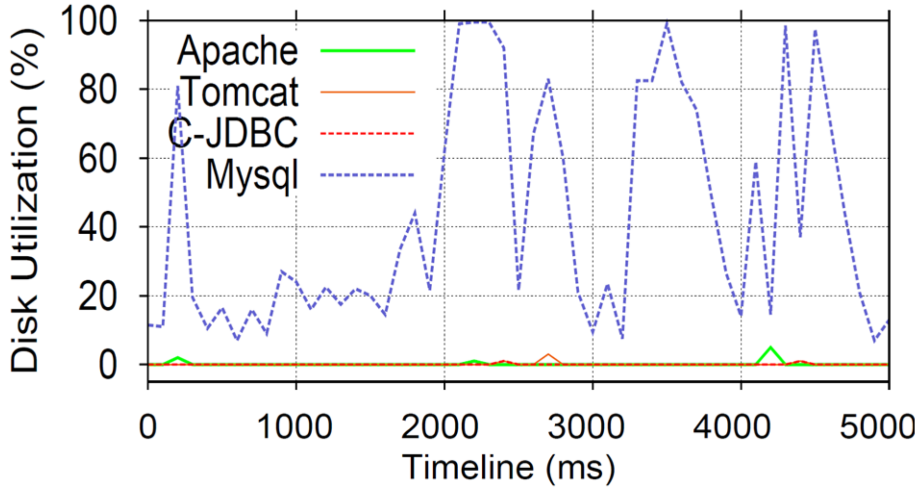


Figure 8: Disk Utilization at different n-tier component nodes in the same period as shown in Figure 2. We observe that disk of Mysql has reached full utilization a couple of times during this period.

3.4 Memory Dirty-Page as the milliBottleneck

A dramatic increase in Point-In-Time response time might be caused by different system components and different system layers. With milliScope, researchers are able to utilize a variety of the fine-grained resource mScopeMonitors and integrate the related data easily. In this section, we show 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 10. After identifying the execution boundary of each request with the event mScopeMonitors and calculating the request queue lengths for each tier in Figure 11, we found these 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 the request queue length of Apache increases, while the request queue length at both Apache and Tomcat increase at the second peak. In the other words, cross-tier queue amplification is observed only at the second Point-In-Time response time peak.

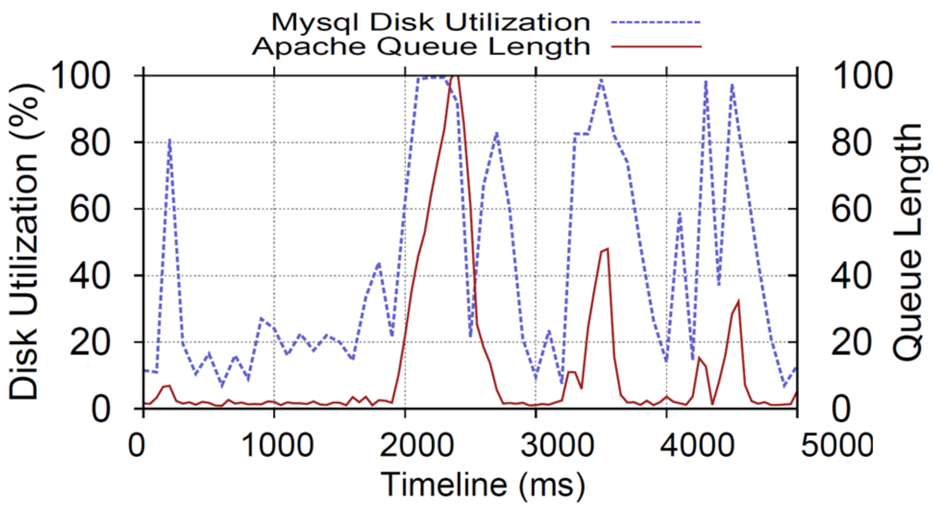


Figure 9: Further investigation for Figure 2 via milliScope. Once the IO of the database tier is saturated because the database flushes logs from memory to disk, the requests at the Apache tier starts queueing. This figure shows disk IO is the milliBottleneck and makes the Point-In-Time response time increase dramatically during the milliBottleneck period.

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 in Figure 12. However, the reason for CPU saturation differs from the previous case study (IO activities), since we do not 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 realize 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 13, the abrupt drop in dirty page cache size correlates with CPU saturation, which suggests that the dirty page recycling on the Apache and Tomcat tiers are the reason for the increasing Point-In-Time response times during these periods.

3.5 Docker for Cross-Tier Queue Overflow

The long tail latency problem arises in distributed systems with tightly-coupled servers using RPC-style request-response communications: Cross-Tier Queue Overflow (CTQO). Recently, researchers have shown that CTQO can be avoided by replacing the server dripping

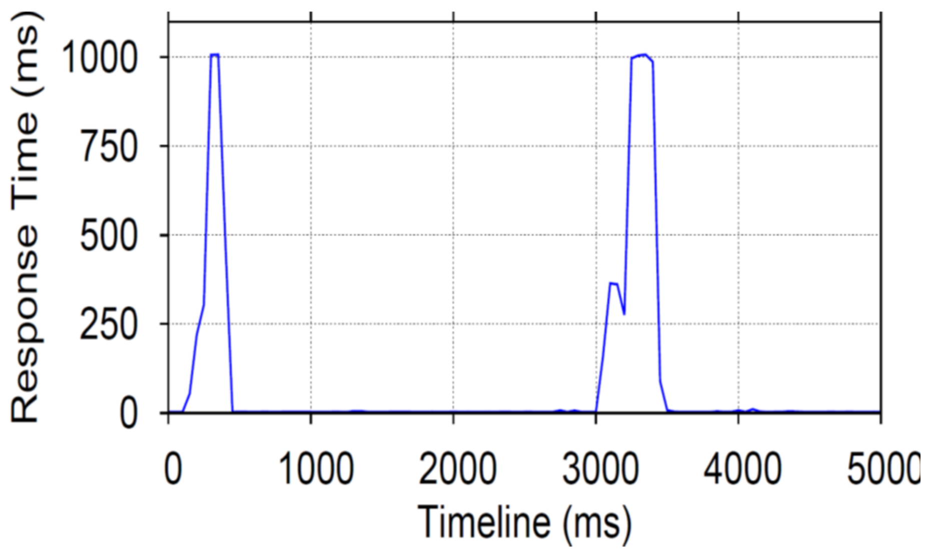


Figure 10: Point-In-Time response time for a n-tier system reaches 1,000 millisecond twice while the average response time is less than twenty millisecond. Although they look similar at the first glance, they are actually caused by different components of an n-tier system.

packets with an asynchronous server [81]. By repalcing all servers with their asynchronous versions, CTQO phenomena can be removed despite workload bursts in moderate average resource utilization. To study if the argument is also validated when the n-tier application is running in virtual machines or containers to share the hardware resource with other applications in the cloud environment, we extend the monitoring scope by introduce the new event and resource mScopeMonitors: Nginx mScopeMonitor and LockStat mScopeMonitor. Concretely, Nginx is an event-based web server, while LockStat monitors the status of locks in the kernel space.

Virtualization technology is popular because it allows multiple applications sharing the computing infrastructures such as computing clouds with little or zero interference. The applications running inside the container or virtual machines recognize nothing about the virtualized environments, while the hypervisor or container daemon is responsible for hardware resource sharing. However, contrary to the above argument, we found evidence showing that the cloud environment imposed by virtualization technology can change the behavior of the application and cause the performance penalty that supposes to be eliminated by event-driven architecture.

Since Apache is the source of VLRT requests in both case described in Section 3.3

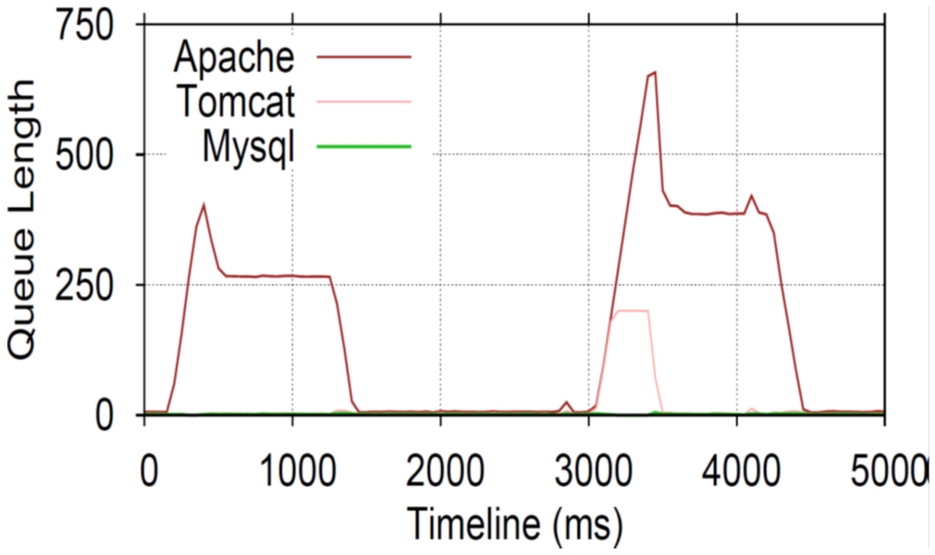


Figure 11: Request queue length for each tier shows an interesting phenomena. During the first peak, only request queue length at Apache increases, but the request queue length at both Apache and Tomcat increase at the second peak.

and Section 3.4, we replace the synchronous Apache with an asynchronous web server, Nginx, to solve the upstream CTQO problem and remove VLRT requests. Furthermore, each component server is running inside a docker container to achieve better scalability. Previous research has proved the Nginx indeed will not drop packets [81] in the dedicated environment. However, our experiments show that asynchronous application still causes CTQO problem because of the synchronous design of container technology, which is Docker in this case.

Since the experiment setup is slightly different from Section 3.2, we describe the detail of the docker experiment here. While consolidation in practice may be applied to any type of application running inside docker container, the focus of our experiment is to demonstrate the synchronous architecture of container technology that will have impact of performance for n-tier application even though it adopts asynchronous design aiming at removing CTQO. Hence, for each physical computing node, we only run one component server (including Nginx, Tomcat, Mysql) in one container. Hardware and software specification is listed in Table 3. Moreover, we enable lockstat in linux kernel to monitor the status of locks in kernel space. In addition, we use ”devicemapper” rather than ”aufs” as storage driver for docker, since using “aufs” as storage driver and lockstat simultaneously will cause kernel panic. We

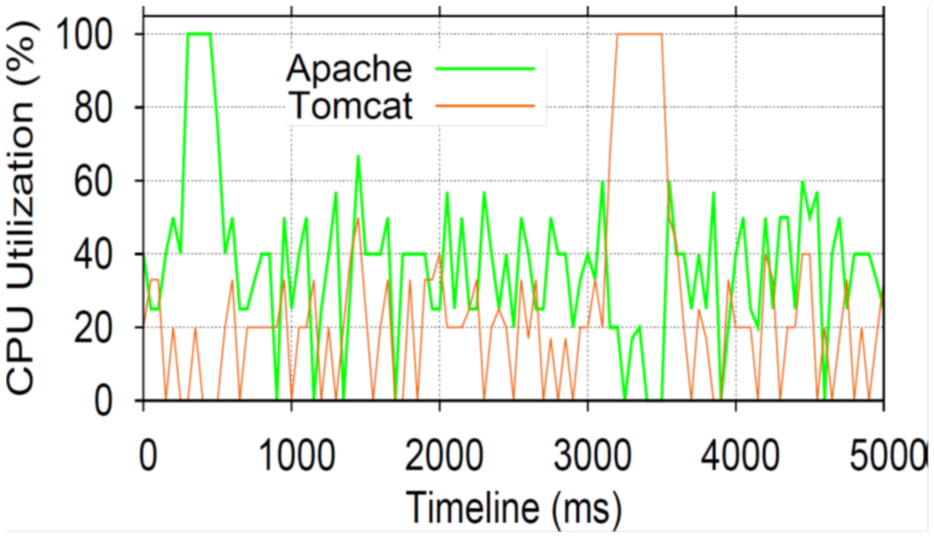


Figure 12: Checking the monitoring data through milliScope, we found CPU utilization of Apache and Tomcat are saturated at the first and the second peak respectively.

adopt RUBBoS benchmark with 6,000 workload executing on browse-only mode.

The experimental results in Figure 14 show that milliBottleneck is observed in a hundreds of millisecond period, in which the Point-In-Time response time for a n-tier system reaches 500 millisecond while the average response time is around twenty millisecond. Moreover, Figure 15 shows the queue length of both Nginx and Tomcat is increasing simultaneously that implies Nginx fails at removing CTQO phenomena. When we overhauled the monitoring data collected from Collectl mScopeMonitors at each component server host, we found the root cause of milliBottlenecks in this period is because the CPU utilization for the four cores of Tomcat reach over 90% as shown in Figure 16. On the other hand, IO utilization for each tier doesn't exceed 75% in the period milliBottleneck happened as shown in Figure 17. Furthermore, the result from Lockstat mScopeMonitor demonstrates that the lock holding time for "RCU_read_lock" at Tomcat has an obvious peak as shown in Figure 18, since the high resource utilization would make the tasks wait for longer time until they get enough resource to finish.

The reason for CTQO phenomena existing is because docker in its current form places any docker commands into a work queue and executes them one at a time sequentially.

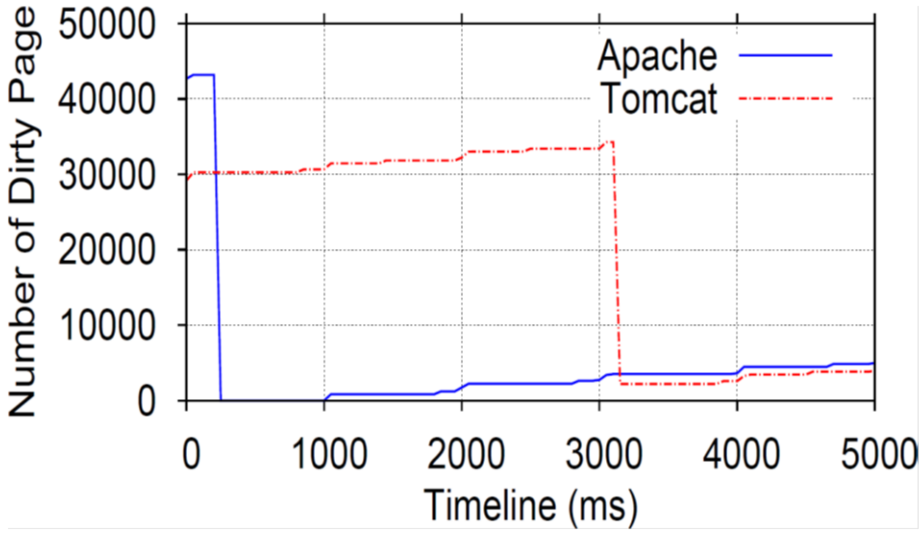


Figure 13: Applying memory subsystem of Collectl mScopeMonitor, milliScope transforms the relative logs, such as number of dirty page, to structured tuples and loads them into mScopeDB. We demonstrate the root cause of CPU saturated is due to dirty page recycling.

In the other hand, docker applies synchronous architecture to perform applications' requests. In this case, a simply asynchronous function call, such as downloading image for example, would incorporating docker command would get backed very quickly but overload the docker work queue, where the other function calls waiting for exectution. We adopt the default parameters setting for linux kernel 4.4, which limits the "TasksMax" system attribute as 512. A request would be blocked if the the threshold was reached and cause the upstream tier queue amplification. From our observation, the Nginx-Tomcat-MySQL configurations with default setting shows that the replacement of the synchronous Apache with the asynchronous Nginx might fail at removing the web server from the Cross-Tier Dependency sequence and still cause the upstream CTQO between Nginx and Tomcat due to the synchronous architecture of virtualization technology. Potential solutions includes increasing the value of "TaskMax" so that it reduces the chance of this situation happening. Systems with high availability requirement might consider running docker in swarm mode, which consists of multiple docker hosts. A running container is as a swarm service and managed by swarm manager, which uses ingress loading balance to automatically assign available resources to service.

Table 3: Summary of docker experimental setup (i.e., hardware, operating system, software).

CPU	Xeon E5530 2.4GHz * 4CPU
Memory	12 GB 1066 MHz DDR2
HDD	SATA, 250GB 7200 rpm Seagate SATA disk
Network I/F	1Gbps
Web Server	HTTPD-2.2.22
App Server	Apache-Tomcat-5.5.17
Connector	Tomcat-Connectors-1.2.32-src
Cluster middleware	C-JDBC 2.0.2
DB Server	MySQL-5.5.19-Linux2.6-i686
Java	JDK1.6.0_27
Monitoring Tools	Collectl
Docker version	1.12.6
Operating System	Ubuntu Trusty 14.04 64-bit
OS Kernel	Linux kernel 4.4

3.6 Related Work

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 [80], Java garbage collection (GC) at the system software layer [79], and virtual machine (VM) consolidation at the VM layer [78] [59]. In this chapter, we utilize milliScope described in Chapter 2 and highlight other two different reasons for the milliBottleneck, database I/O activities and memory thrashing respectively.

The factors affecting database transaction performance have been studied and steady progress has been made in resolving these issues [21, 23, 25]. Commit, one of the most important functions for preserving the ACID property of transactions, has been shown to increase database processing time because of the incurred disk access time [30]. Remesh et al. [34] investigated the distributed real-time database system performance of commit

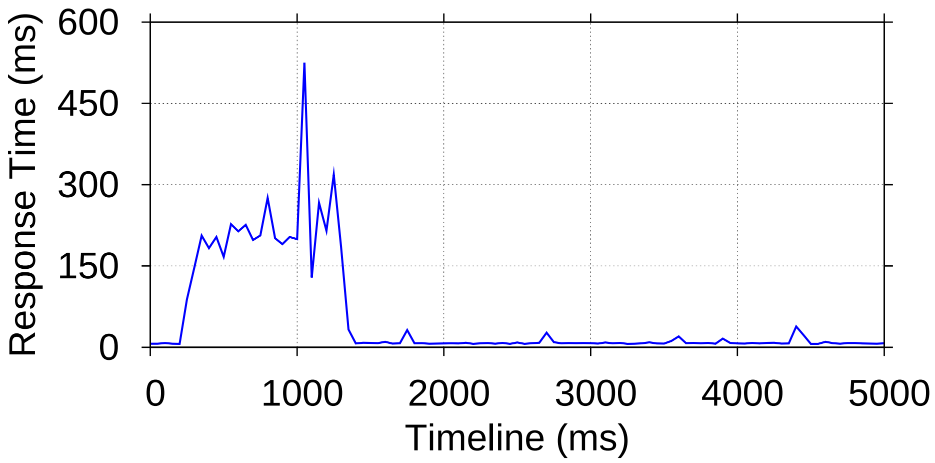


Figure 14: Point-In-Time response time for a n-tier system reaches 500 millisecond while the average response time is around twenty millisecond. Each component server is running inside a docker container and located at different physical machines.

protocols, and proposed the OPT protocol to improve the performance of commit over standard algorithms, such as 2PC and 3PC. Lowell et al. presented a system that reduces the transaction overhead by replacing the standard redo log with recoverable memory [50]. Recently, with SSD (Solid State Drive) storage systems, transaction throughput has been improved by a factor of 3.5 over traditional hard disk drive systems [44]. Recently, C. Lai et al. have shown that single group commit, fsync, would incur large disk I/O in a short time and degrade the system performance in cloud environment [43].

Memory thrashing due to page faults a well-known issue to system community. For example, Banerjee et al. conducted comparative analysis for various memory over-commitment methods which are implemented on popular hypervisors, including ESX, KVM, Hype-V and Xen [74]. Besides, through sub-page level page shring with pathing, Gupta et al. demonstrated the better use of host memory in Xen [33]. Moreover, Junhee et. al. showed that in typical cloud virtualizatin environment, the memory thrashing can occur without memory over-commitment and it can significantly limit the virtual machine consolidation ratio [60]. In this chapter, we proved that even with moderated system utilization, the mechanism of memory thrashing, the number of dirty page reaches the threshold so the garbage collector is executed, would cause the long milliBottleneck and increase the point-in-time response time in a short period.

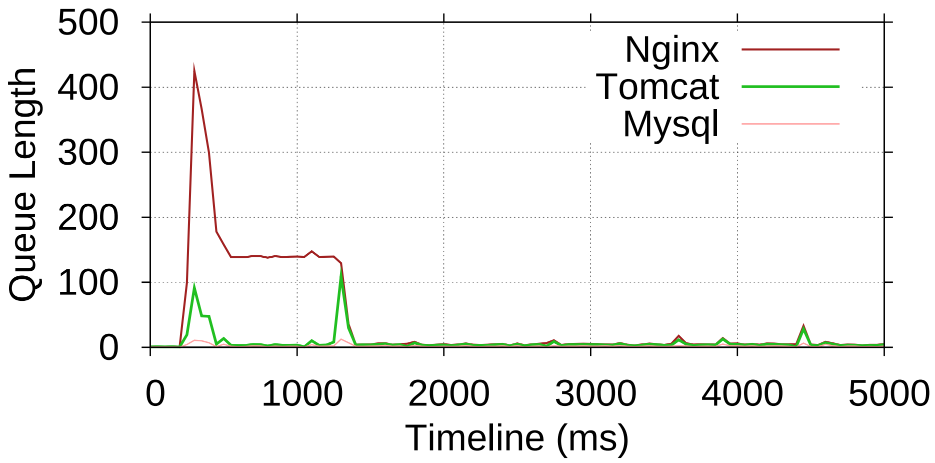
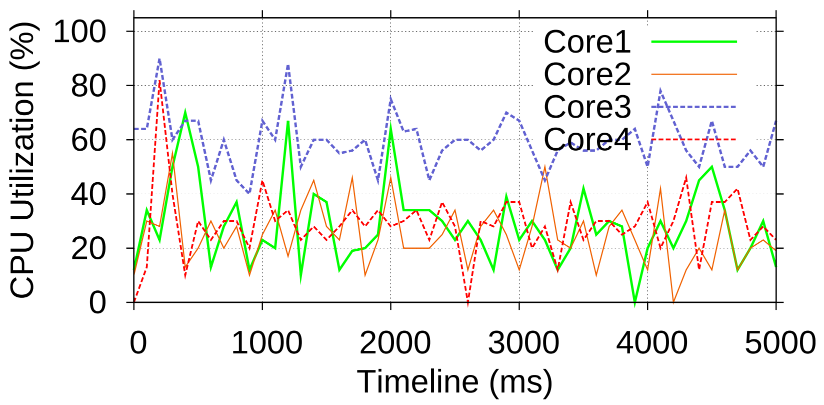


Figure 15: Previous researches showed the asynchronous architecture, such as Nginx, is able to removed the Cross-Tier Queue Overflow (CTQO). However, CTQO phenomena is observed here since docker is still synchronous architecture. During the period that Point-In-Time response increasing, the request queue length at both Nginx and Tomcat increase.

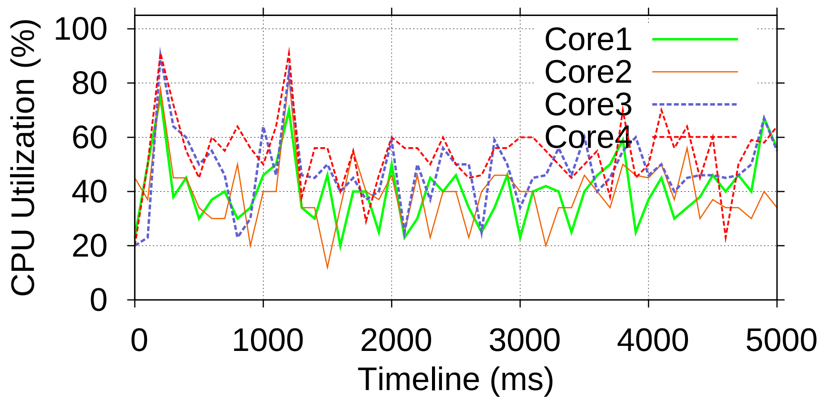
The performance variations of computing nodes makes the processing of a normal request to be unexpectedly long. According to an Amazon report [1], an increase of 100 milliseconds in system latency can lead to a 1% loss in sales. Isolating the root cause of these bottlenecks is challenging because of their fleeting nature and the large number of potential causes [80] [78]. Several researches have proposed solutions to the unexpected very long response time (VLRT) requests without identifying the specific sources. For instance, Dean et al. used service replication to bypass tail latency of interactive application in Google [22], while C3 duplicated the adaptive selection scheme in storage servers to avoid VLRT requests [69]. With milliScope, researchers and system administrators are able to find the root cause of the milliBottlenecks and tackle the long tail problem through the corresponding solutions.

3.7 Conclusion

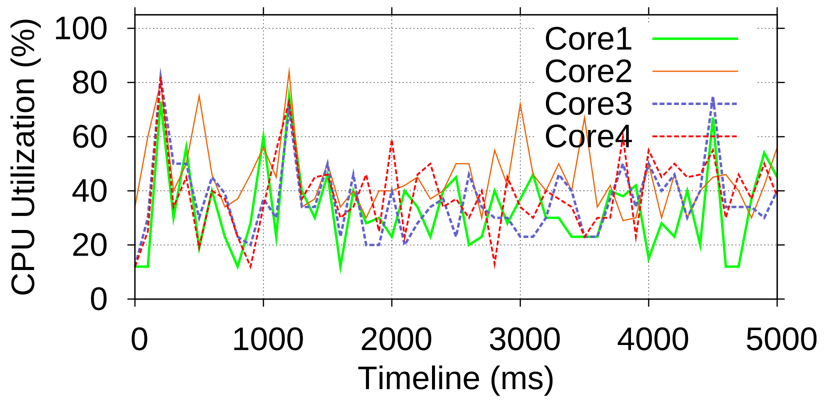
Except the milliBottleneck which have been shown in previous researches, we provide two new illustrative scenarios in Section 3.3 and Section 3.4 respectively, in which milliScope accomplishes each of the following activities: collects data from the event mScopeMonitors and the resource mScopeMonitors, transforms the native logs into structured data through mScopeDataTransformer and loads it into mScopeDB. With milliScope, we are able to “scale the mountain” of native log that generated by different mScopeMonitors, and eventually look for the root cause of observed performance anomalies. In both scenarios, we discover the milliBottlenecks that cause the VLRT requests, and we demonstrate milliScope makes no assumptions about the origin of bottlenecks. This is exhibited by highlighting the different reasons for the milliBottlenecks in the scenarios, database I/O activities and memory thrashing respectively.



(a) CPU utilization of each core for Nginx.



(b) CPU utilization of each core for Tomcat.



(c) CPU utilization of each core for Mysql.

Figure 16: Checking the monitoring data through milliScope, we found CPU utilization of Tomcat are increasing obviously for all the four cores in this period.

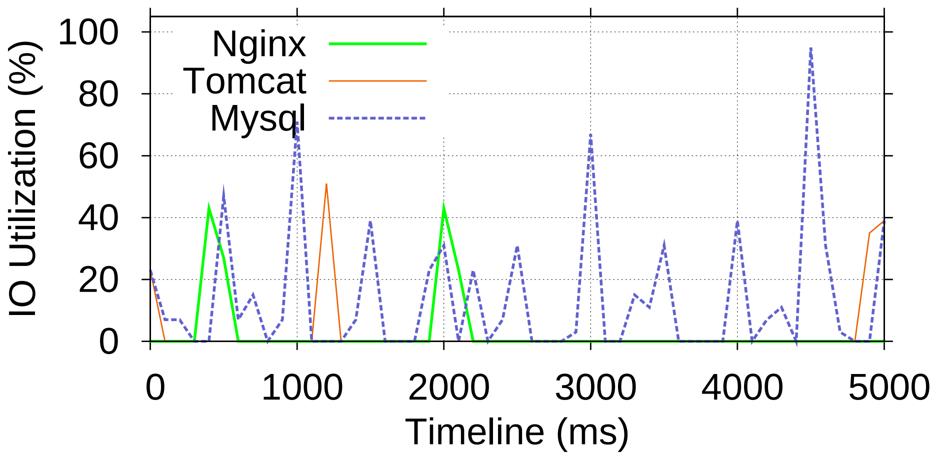


Figure 17: On the other hand, IO utilization for each tier doesn't exceed 75% in the period milliBottleneck happened.

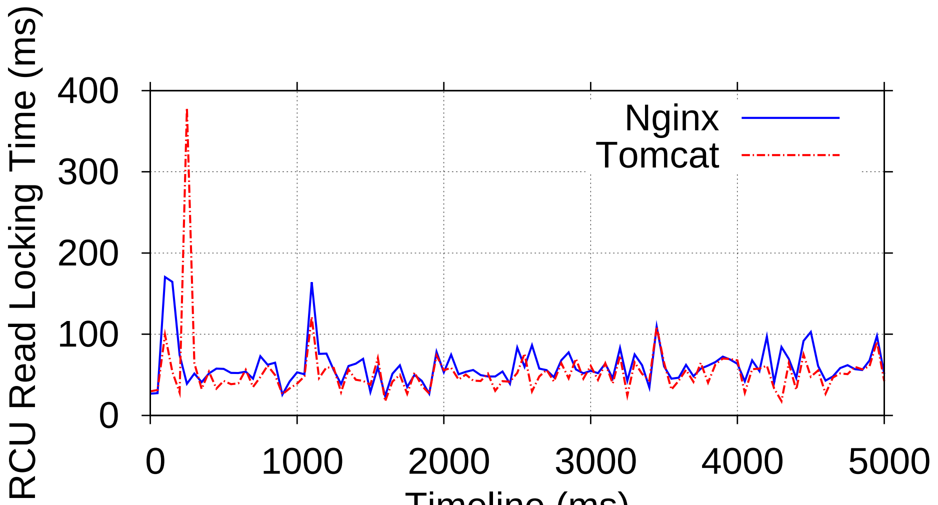


Figure 18: To extent the monitoring scope, we apply lockstat mScopeMonitor to check the status of lock in the kernel space. The result shows the lock holding time for "RCU_read_lock" at Tomcat has an obvious peak in this period. This demonstrates Point-In-Time response time increases is because the increasing of CPU utilization at Tomcat causes the Cross-Tier Queue Overflow.

CHAPTER IV

MORE EVALUATIONS FOR MILLISCOPE

In the previous chapter, we have shown that milliScope is able to scale the mountain of native logs, which are generated by the event mScopeMonitors and resource mScopeMonitors, to detect the milliBottlenecks successfully. In this chapter, we would like to study the characteristics of milliScope more. First, we discuss the impact of the overhead of the monitoring tools in Section 4.1, and we describe our experiment setup in Section 4.2. We show concrete experimental evidence of the accuracy through comparison between the monitoring data of milliScope and the monitoring records of Fujitsu SysViz [41] in Section 4.3. After that, we provide the detail of overhead of milliScope to show it is a light weight fine-grained monitoring tool in Section 4.4. Furthermore, we demonstrate its flexibility and extensibility in Section 4.5 and Section 4.6 respectively. Section 4.7 provides an overview of the related work in this area. Finally, Section 4.8 concludes this chapter.

4.1 Introduction

End-to-end tracing tools have been used to analyze the distributed system performance for a wide variety of use cases. For instance, they can be used to identify suspicious workflows and abnormal latency. They are also helpful for improving the system's resource efficiency and ultimately the end user experience. According to an Amazon report [1], an increase of 100ms in system latency can lead to a 1% loss in sales. Conversely, improvements to their system's performance can lead to higher business profits; however, this occurs at diminishing returns in part because of the extremely difficult proposition of the root cause of the milliBottlenecks among the component servers in their distributed systems with a negligible overhead.

Currently, coarsed-grained tracing tools play key roles in performance debugging and optimization, because they support system administrators and programmers in diagnosing bottlenecks in distributed systems, which ultimately enables these stakeholders to overhaul the system's request processing. While black box approach relies on the assumption for the specific system models in mind, white-box approach is more generic for the dynamical cloud environment. However, although these tracing tools in white-box category provide different features for performance debugging, they all adopt sampling approach to reduce monitoring overhead, which limit their availability for milliBottleneck detection.

The milliBottlenecks (e.g., JVM garbage collection and Intel SpeedStep) have been proved to be the reason for significant performance loss, but the study of the milliBottleneck has been hampered due to the short lifespan [77]. Monitoring tools which sample at time intervals measured in seconds or minutes are not capable of detecting these kind of bottlenecks from Sampling Theory. Fugitsu SysViz [41] adopts machine learning techniques and statistical regression to reconstruct the entire trace of each transaction executed in the system, but it's lack of scalability because all the servers have to be connected to passive network tracing support network switch.

In the previous chapters, we have presented milliScope, the first millisecond granularity, software-based resource and event monitor for distributed systems. milliScope is an end-to-end tracing tool that is the most closely related to other previous excellent ones, such as Dapper [67], Magpie [12] and X-Trace [28]. Instead, milliScope aims to provide each request's execution footprint in a distributed system while imposing negligible overhead. Instead of extending an existing RPC library, milliScope just adds a global identifier to each request. As the request flows through the distributed processing system, the added identifier is propagated across the systems' component servers. By using very simple instrumentation at critical points in the system's infrastructure, such as the points connecting the upstream and downstream tiers, and capturing this minimal information in system logs, milliScope delivers on three of the most important features of tracing:

- Precise: milliScope reconstructs the explicit causal relationship using a global identifier and associated timestamps.
- LightWeight: milliScope keeps the tracing results in memory and integrates this information into the existing log files of component servers.

Table 4: Summary of experimental setup (i.e., hardware, operating system, software).

CPU	2 Quad Q9650 3GHz * 4CPU
Memory	16GB
HDD	SATA, 7,200RPM, 500GB
Network I/F	1Gbps
Web Server	HTTPD-2.2.22
App Server	Apache-Tomcat-5.5.17
Cluster middleware	C-JDBC 2.0.2
DB Server	MySQL-5.5.19-Linux2.6-i686
Operating System	RHEL Server 6.3 64-bit
OS Kernel	2.6.32-279.22.1.e16.x86_64

- Scalable: milliScope interleaves the tracing code into each component server, hence inheriting the same degree of scalability as the underlying distributed system.

The main focus in this chapter is that we would overhaul the characteristics of milliScope to see if it is able to achieve highly precise, lightweight, and scalable. Specifically, we validate the precision of milliScope with Fugitsu SysViz [41], a passive network tracing tool. In comparison to other white-box monitoring tools such as Zipkin [5], milliScope introduces only 1% to 3% of CPU overhead into the system-under-study, without sampling, and it helps diagnose response time anomalies caused by milliBottlenecks. We also show that milliScope is easily plugged into other web applications by adding instrumentation in only a few lines of source code.

4.2 Experimental Setup

We execute the RUBBoS benchmark [4] with the event mScopeMonitors and SysViz running at the same time on their own n-tier systems (but in the same configuration) and compare the monitoring results of each tier as determined by these two monitoring tools respectively. The configuration we use for validation appears in Figure 19, while the hardware and software specifications are listed in Table 4.

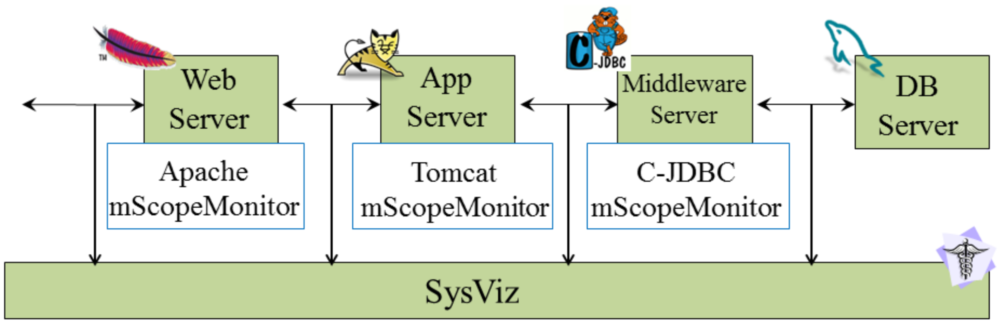


Figure 19: Hardware configuration for validation experiment. The event mScopeMonitos are deployed in each component server, while the SysViz machine is connected to the network switch for passive network tracing.

4.3 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 [41]. SysViz is able to reconstruct the entire trace of each transaction executed in a system based on the interaction messages collected through network taps or network switches that support passive network tracing. Additionally, we use RUBBoS, a standard n-tier benchmark [4], 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 workload represents the number of concurrent users. Each experimental trial is running for 7 minutes.

In this section, we show the queue length, an important metric that can be derived from the request flow tracing data [77] for each tier at workload 1,000, 8,000 and 9,000 as depicted in Figure 20, Figure 21, and Figure 22 respectively. 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.

4.4 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 [4] benchmark when the event mScopeMonitors are enabled on each of the component nodes of the underlying n-tier system. Figure 25 shows that there is almost no difference in the system throughput despite the fact that the event mScopeMonitors are enable or not. 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 23 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 the 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 Figure 24 that 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 4.2. Taking Figure 23, Figure 24, and Figure 25, 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 4, 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.

4.5 milliScope Flexibility & Availability

Different resource mScopeMonitors target at different system components and provide different numbers of variables for monitoring as shown in Table 5. Moreover, they usually allow users to customize their reports, so hundreds of possible log formats might be generated. milliScope uses mScopeParsers to enrich the semantics and syntax of these arbitrary log files to infer a database schema. The unstructured log files are converted into some structured tuples through a multi-stage approach.

Table 5: Number of variables monitored by default. Users are allowed to customize the report format, so each resource mScopeMonitor might generate hundreds of different formats. milliScope handles the variety of log format through mScopeParsers as mentioned in Section 2.2.

Monitoring Tool	Number of Monitoring Variables
SAR	8 * (Number of CPU cores)
IOStat	12 * (Number of disks)
Collectl Memory	10
Collectl Network Subsystem	13 * (Number of network interfaces)

Another reason for the study of very short bottlenecks had been hampered is that fine-grained monitoring requires tracing all of the requests without sampling and it produces the voluminous amounts of data that is hard to process and analyze. In the past year, we have executed more than four hundred of experiments in which the event mScopeMonitors generate about fifty million entries overall cross different number of files depending on the experiment configurations as shown in Figure 26. milliScope demonstrates its flexibility and availability by parsing the voluminous of data and storing in mScopeDB, so that researchers are able to select the relevant monitoring records through structured query languages like SQL language.

4.6 Extensibility

Our approach is fully realized when migrating milliScope to new benchmarks. With the event mScopeMonitors already deployed onto component servers, only a few pieces of source code need to be instrumented. A portion of the modifications for one benchmark is illustrated in Figure 27. Specifically, we apply milliScope to the RUBBoS and RUBiS benchmarks. Each of these benchmarks contains more than 6,000 lines of servlet code; however, less than 300 lines of code have to be added to make milliScope work with these applications.

To extent our code specialized methodology to more component server, we currently rely on manual instrumentation by sophisticated programmers. However, section 2.4 has demonstrated the number of lines to modify the original source code of component server is usually less than 50 lines. With a few modification, component servers will be able to record the extra metadata and utilize their existing logging facilities and achieve fine-grained monitoring eventually. The instances shown in Section 2.4 implies that the need of source code specialization by a programmer can be potentially eliminated by using systematical and automated solution. One possible technique to address this issue would be annotation language-level constructs. For example, programmers might be able to use aspect-oriented programming in the form of AspectJ pointcuts to decorate targeted code with our measurement instrumentation. We would like to see more related research in this field in the future.

4.7 Related Work

Sampling is the most popular technique used by the end-to-end tracing infrastructures to prevent performance degradation due to runtime and storage overhead [16] [27] [67] [66] [71]. For example, Dapper uses sampling to capture 0.01% of all trace points, that reduces the overhead from 1.5% to 0.06% and from 16% to 0.20% in terms of throughput and response time respectively [67]. Moreover, sampling is also helpful to limit the sizes of analysis-specific data structure even when the trace-point records do not need to be persisted in online analysis scenario. Generally, the methods to determine if the trace points would be sampled could be categorized as these three options: head-based coherent sampling, tail-based coherent sampling, and unitary sampling [16].

Head-based coherent sampling makes a random decision for the entire workflow when the requests enter the system, and it propagates the metadata with workflows along to indicate whether to collect their trace points. It is popular and used by many existing tracing implementations due to its simplicity [66] [16] [27]. Since the effective trace-point-sampling percentage is almost always much higher than the workflow sampling percentage, head-based coherent sampling is unable to reduce runtime and storage overhead for monitoring tools that perserve submitter causality. For instance, although Stardust [71] only adopts sampling rate of 10%, 97% trace points are sampled in their testbed distributed system, Ursa Minor [6], since the system contains a cache that aggregated 32 items at a time at the entry point.

Tail-based coherent sampling is different from the head-based coherent sampling by the sampling decision made at the end of workflows rather than at their starting. The advantage of delay the sampling decision is that the monitoring tool is able to choose only to collect anomalous requests through examining their properties, such as response time. However, the monitoring data for the requests have to be stored somewhere before the sampling decision is made for them. A hybrid approach is also proposed in which the monitoring tool use head-based coherent sampling, but it also records the executed tracing points in per–node circular buffer. By doing so, the monitoring tool could backtrack and collect the data for non-sampled workflows that appear anomalous.

Unitary sampling relies on developers setting the trace-point sampling percentage directly, and it makes sampling decision at the level of individual trace points. Developers must decide how to sample requests at the trace points as well as how many of them would be sampled. The sample ratio is often between 0.01% and 10% for most of monitoring infrastructure [16] [27] [67]. Besides, Sigelman et al. proposed an adaptive approach to capture a set rate of trace points and dynamically adjusts the workflow sampling percentage [66], such as five hundred trace points per second or one hundred workflows per second.

However, sampling methods have to be avoided for fine-grained monitoring tool, such as milliScope, because it might overlook the milliBottleneck and the very long response time (VLRT) requests. Meanwhile, black-box monitoring approach is not capable of milliBottleneck detection because it relies on pre-built analytical models and is limited to specific workflows [12] [35] [39] [70] [20] [86]. Comparing with other approaches, milliScope imposes negligible overhead by leveraging the native logging infrastructure accompanying each component server. Each request receives a unique identifier that accompanies the request as it 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.

4.8 Conclusion

In this chapter, we overhaul the detailed characteristics of milliScope, the the first millisecond granularity software-based resource and event monitor for distributed systems. Through multiple evaluation, we demonstrate that milliScope has both acceptable performance (low overhead at high measurement frequency) and high accuracy when compared to other firmware monitors. Concretely, milliScope introduces only 1% to 3% CPU overhead without adopting sampling, but it is able to record up to two times in terms of the data size comparing with unmodified component servers. Moreover, we validate the accuracy and lightweight characteristics of the event mScopeMonitors and demonstrate the availability and flexibility of millisScope through several evaluations. W also showed the flexibility of milliScope by importing it into other web application benchmarks with few lines of source code modification. Overall, milliScope is an important contribution towards performance debugging and milliBottleneck detection of complex n-tier applications in cloud environments.

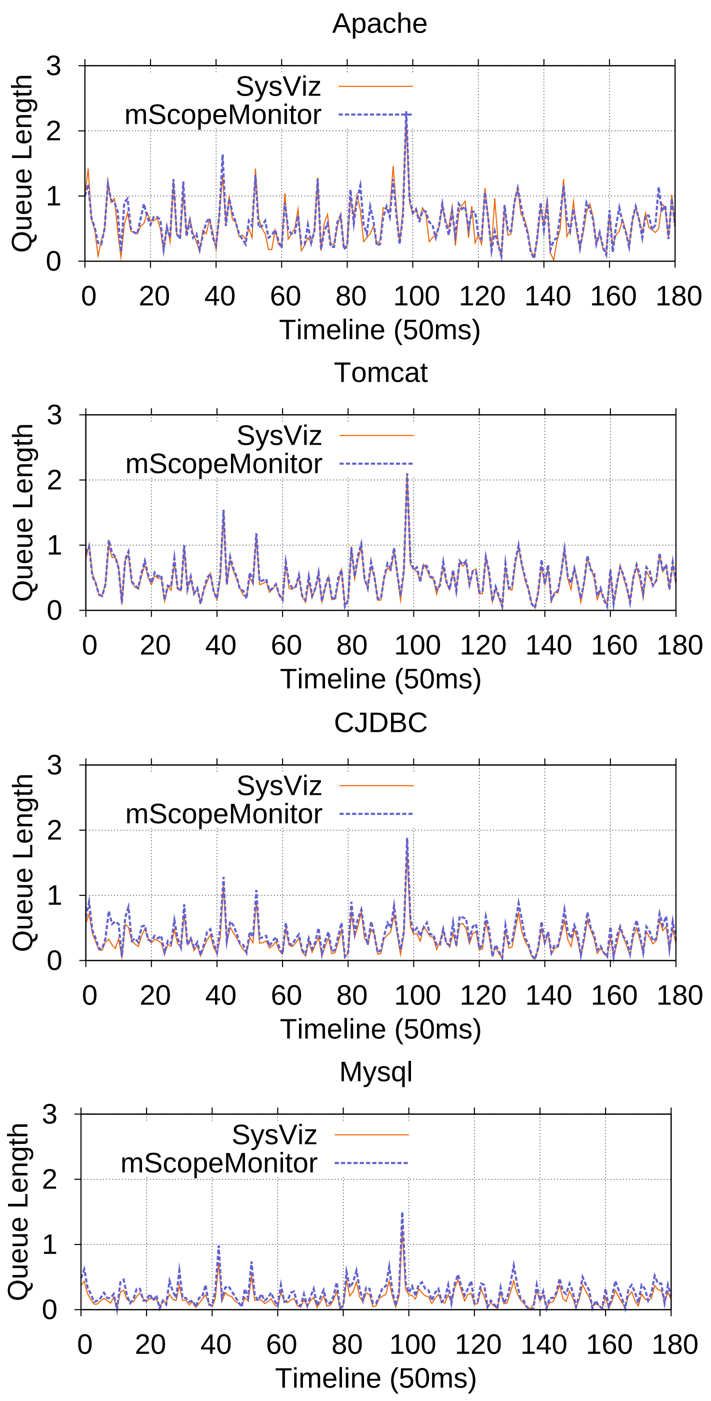


Figure 20: Queue length comparison at workload 1000 between SysViz and the event mScopeMonitors among n-Tier systems, including Apache, Tomcat, CJDBC and MySQL. The event mScopeMonitors' results are very similar to SysViz's, which demonstrates the accuracy of the event mScopeMonitors.

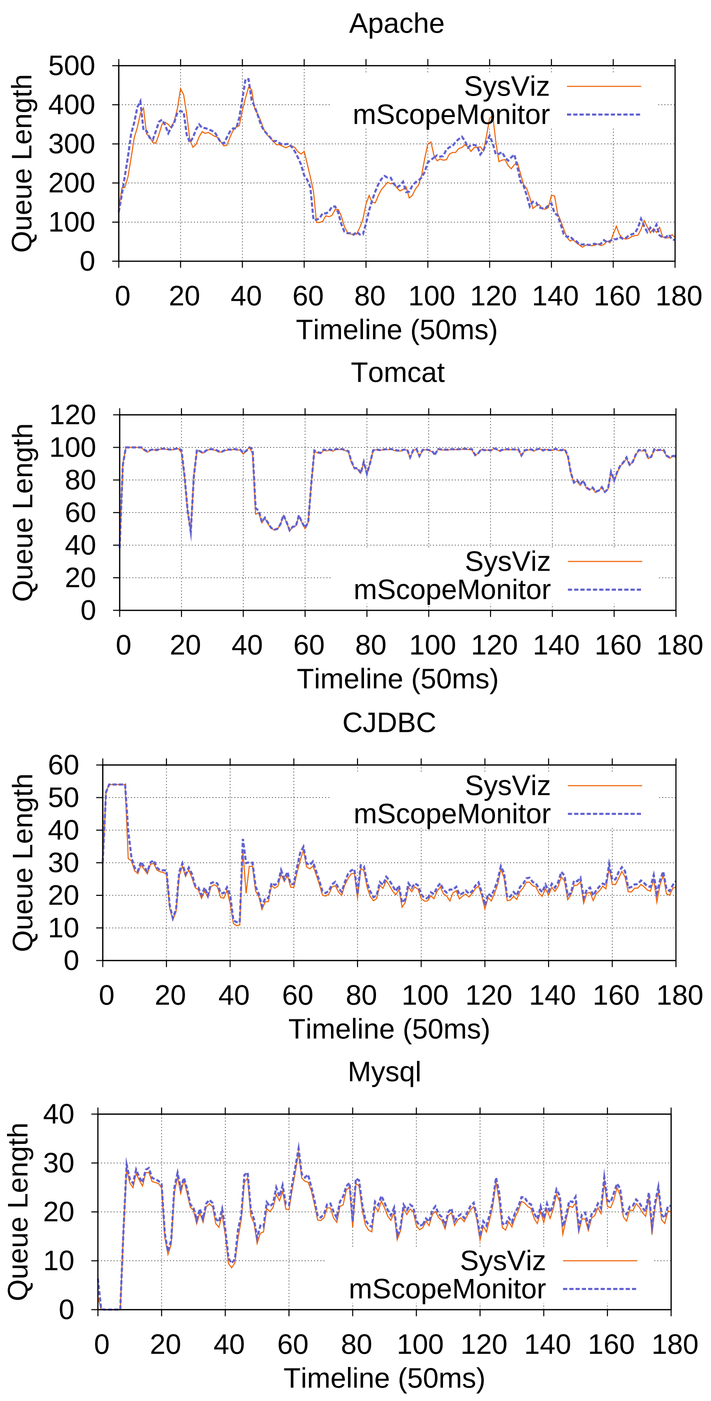


Figure 21: Queue length comparison at workload 8000 between SysViz and the event mScopeMonitors among n-Tier systems, including Apache, Tomcat, CJDBC and MySQL. The event mScopeMonitors' results are very similar to SysViz's, which demonstrates the accuracy of the event mScopeMonitors.

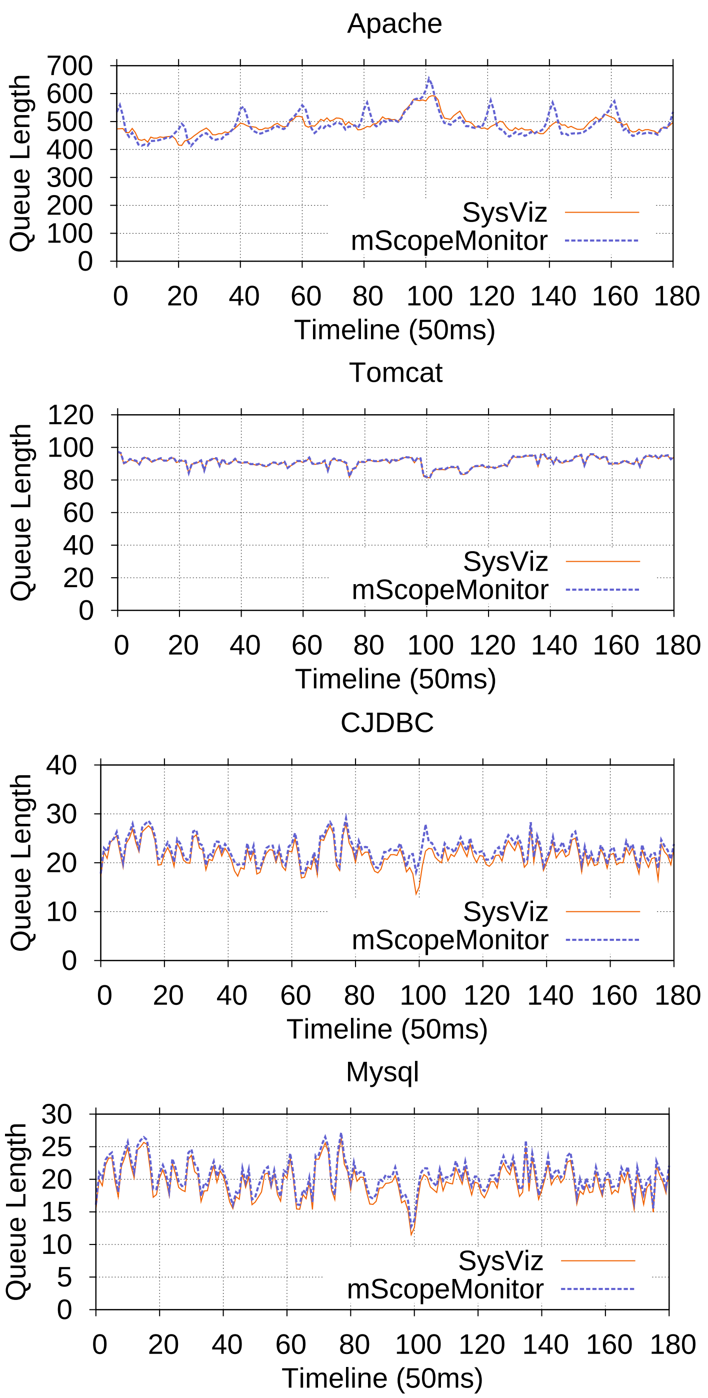
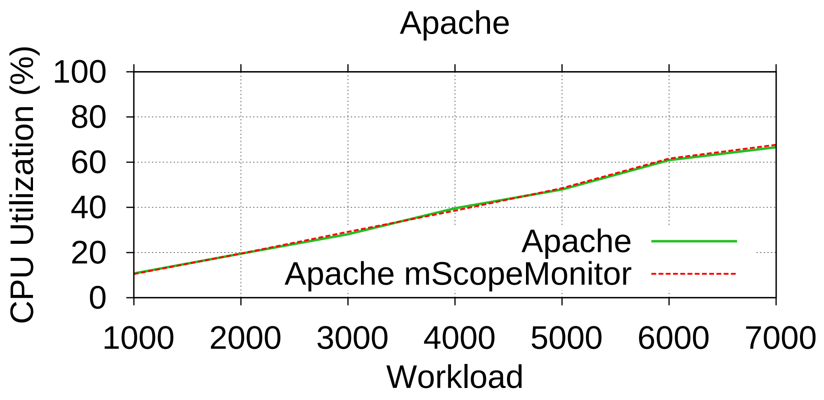
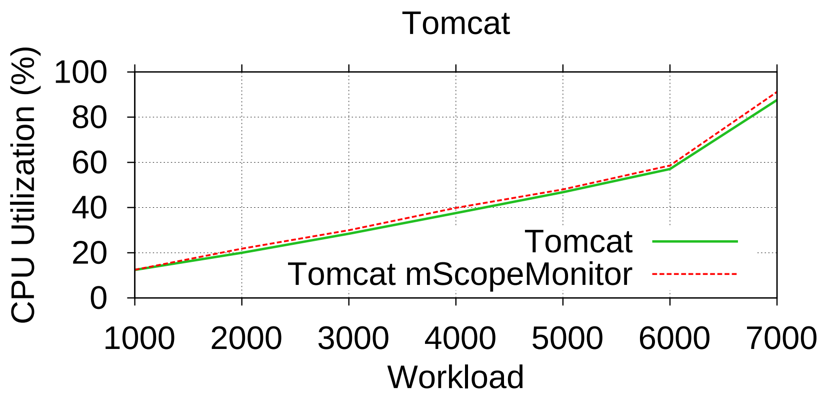


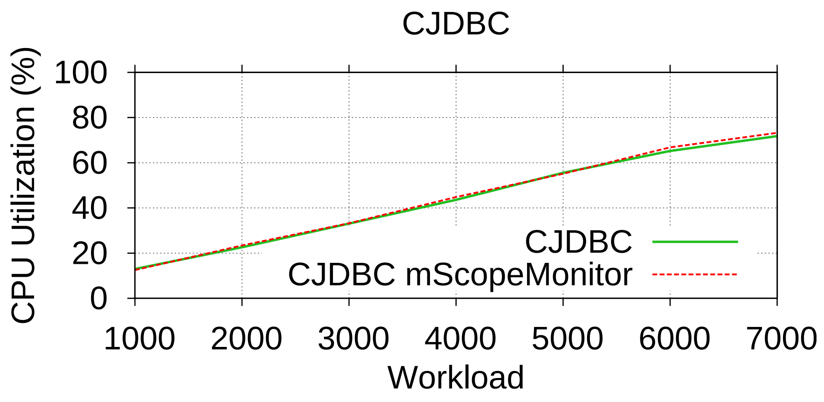
Figure 22: Queue length comparison at workload 9000 between SysViz and the event mScopeMonitors among n-Tier systems, including Apache, Tomcat, CJDBC and MySQL. The event mScopeMonitors' results are very similar to SysViz's, which demonstrates the accuracy of the event mScopeMonitors.



(a) Overhead is about 1%.

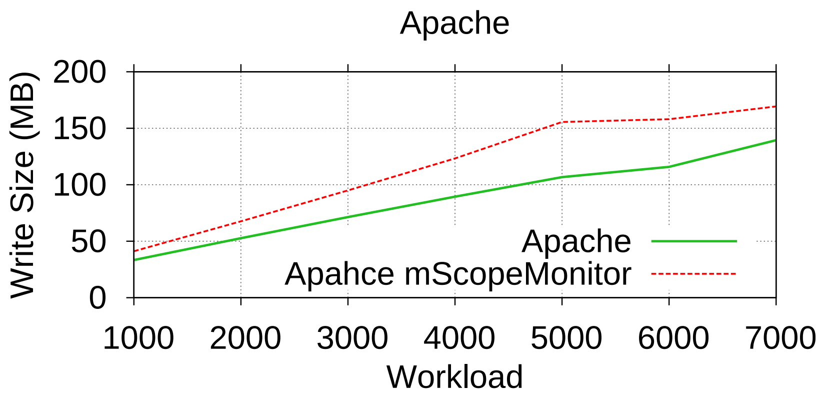


(b) Overhead is about 3%.

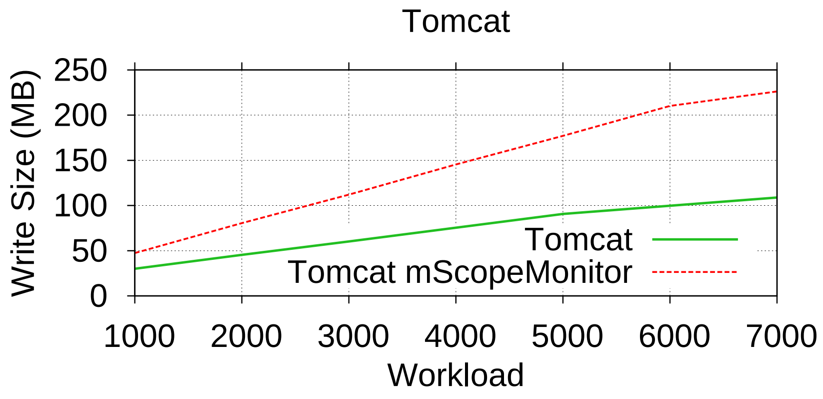


(c) Overhead is about 1%.

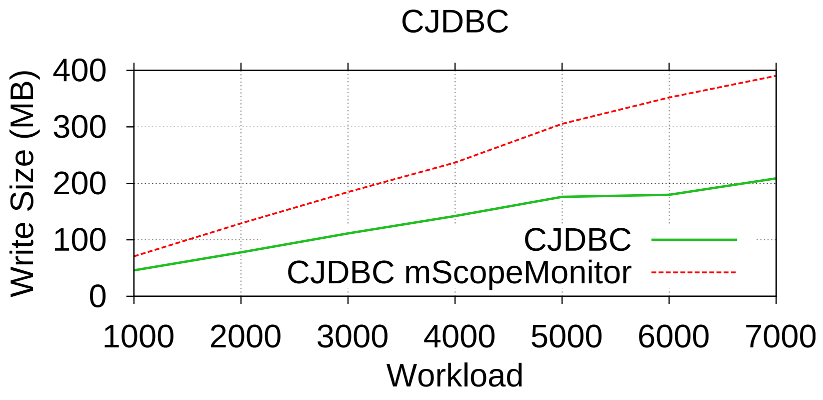
Figure 23: Compared to unmodified servers, the overhead for event mScopeMonitors are increase 1% to 3% CPU utilization.



(a) Write size increases around 33%.

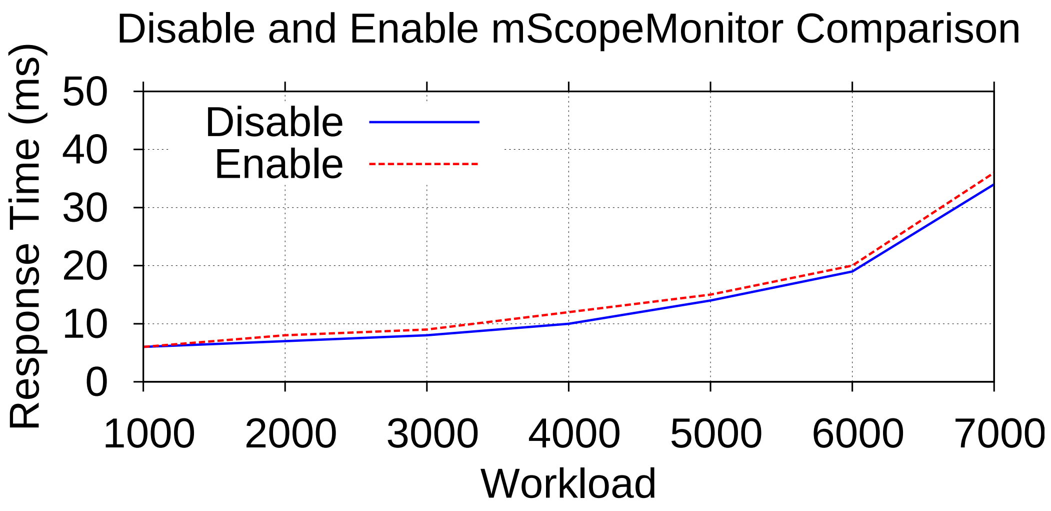


(b) Write size increases around 50%.

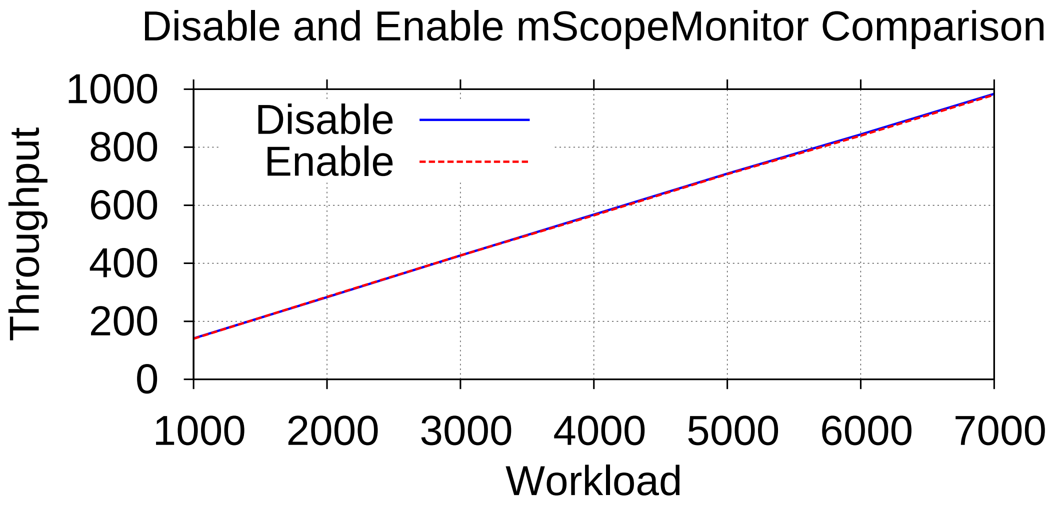


(c) Write size increases around 50%.

Figure 24: Compared to unmodified servers, the aggregated disk write size for event mScopeMonitors are up to two times.



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



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

Figure 25: 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.

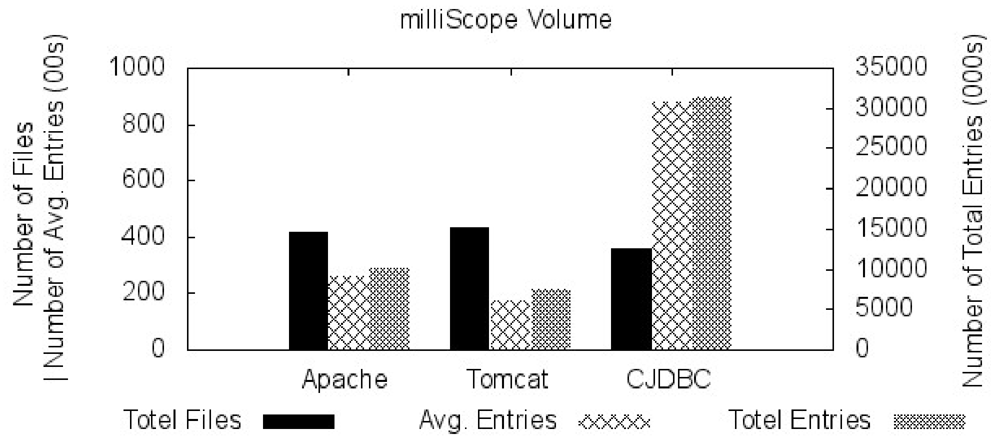


Figure 26: Number of files and entries generated by event mScopeMonitors among more than four hundred experiments. milliScope demonstrates its flexibility and availability by handling voluminous logs in varied formats.

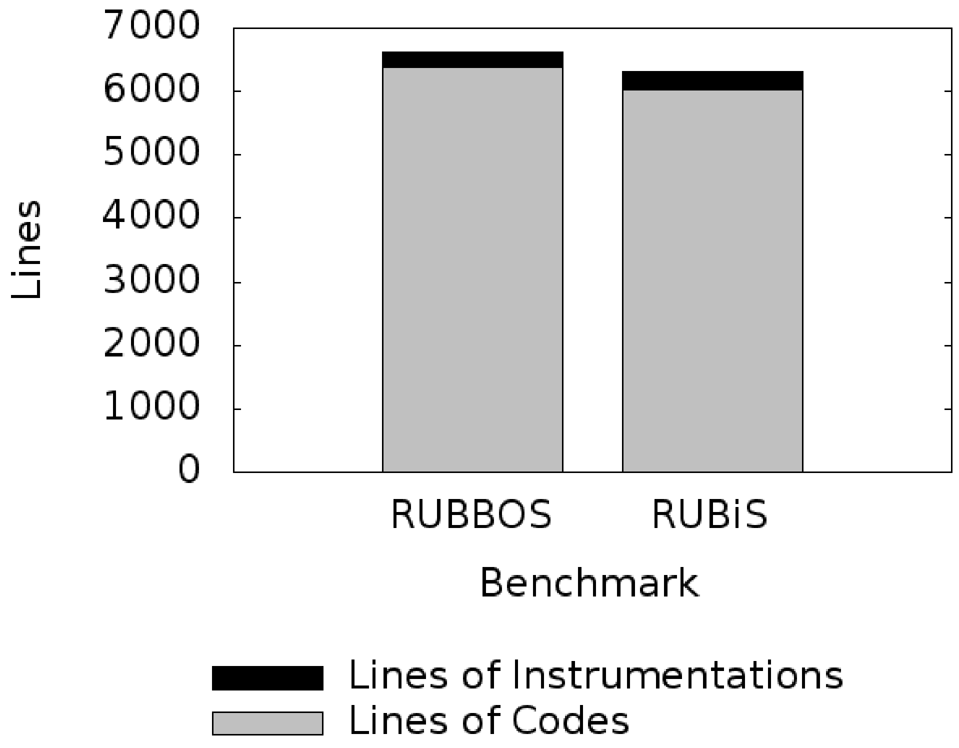


Figure 27: Lines of instrumentation to apply milliScope to RUBBoS and RUBiS benchmark. With more than 6,000 lines of source code for each benchmark, milliScope inserts less than 300 lines for each.

CHAPTER V

RELATED WORK

Many end-to-end tracing implementations insert and propagate metadata (e.g., an ID) to create correlation among individual trace points. For example, Pip [63] aims to detect incorrect behavior by finding rare paths or those that differ greatly from expectation, while Whodunit [16] tracks and profiles transactions flowing through system components to identify latency-inducing components and interference from concurrent transactions. Quanto [26] uses resource attribution by power state, energy metering and causal tracking activities in a distributed-embedded system. Moreover, Stardust [71] creates queuing models to capture workload summaries for performance prediction. Recently, Dapper [67] and Zipkin [5] provide application-level transparency by restricting the instrumentation to common libraries. Pivot Tracing [51] is a monitoring framework that enables users to obtain system metrics at runtime through dynamic instrumentation and causal tracing techniques. To reduce the runtime overhead, most tracing infrastructures, which rely on metadata propagation techniques, adopt sampling to collect relatively small numbers of tracing points or workflows. milliScope, the event mScopeMonitors in particular, inserts a request ID as well, but it captures the entire request execution map without the need of sample.

A few implementations establish causal relationships among variables that are exposed in custom-written log messages. Magpie [12] adopts this approach by not only recording the path of each request but also its resource consumption, while ETE [35] addresses long response time transactions and their associated components by late binding events to transaction definitions. Since these schema-based approaches delay determining the causal relationship in systems until all logs are collected, it's incompatible with sampling and less scalable than a metadata-based propagation approach. Recently, Mystery Machine implements a measurement interface in each component server and uses the output to reconstruct the execution flow across all of these traces [20], while 1prof attempts to leverage existing log messages of systems to extract meaningful performance information, i.e., how to parse them, and where they occur in the execution flow of a system [86].

Several end-to-end request-flow tracing systems have been built in previous research for anomaly detection and performance correction. Magpie [12] and Pip [63] aim at identifying anomalous requests which either have long response times or incorrect behavior by finding rare paths that differ greatly from expectation, while Spectorscope [66] identifies anomalous requests by comparing request-flows between "problem" period and "non-problem" period. Instead of building model of the workload and adopting statistics analysis, Dapper [67] provides low overhead application-level transparency by using sampling and restricting the instrumentation with common libraries.

Black-box implementations use statistical regression analyses to reconstruct causality without modifying traced systems. Draco is a diagnosis engine, which operates by correlating variables from pre-existing logs to address chronic problems, which are often ignored due to the small number of affected users [39]. By making assumptions about programming patterns, vPath discovers the request-processing path observed by monitoring thread and network activities [70]. Recently, Mystery Machine [20] constructs the potential hypotheses about system behavior through large number of pre-existing component logs, and it confirms these hypotheses by the empirical observation on the target system. SysViz [41] can reconstruct the entire trace for each transaction at sub-second levels, making very short bottleneck detection possible [77], but it requires its servers to be connected to network switches, which support passive network tracing. Although this method incurs low-overhead and does not require software modification, it is limited to the specific workflows since it relies on a pre-built analytical model.

These end-to-end monitoring tools with different approaches have been proven useful for many important use cases, including anomaly detection [12] [18], diagnosis of steady-state correctness [18] [27] [28] [63] [66] [67], system performance profiling [16] [67], and resource usage attribution [26] [71]. With the emerging of large and complex modern distribution services, like Google File System [31] and Bigtable [17], more and more industry implementations, such as Google's Dapper [67], Cloudera's HTrace [2] and Twitter's Zipkin [5] for instance, have been built to captures the detailed causally-related activities of the application within and among the components of a distributed system. End-to-end monitoring tools have become the essential parts to provide the advanced analysis of request activities in cloud environments.

Though these detecting anomalous requests some tools to provide very useful hints to diagnose performance problems, they may overlook the response time fluctuations in high resource utilization scenario [76] or fail at transient bottleneck detection [77] due to the granularity limitation. From the sampling theory, due to the short lifespan, the phenomena only can be detectable when the monitor tool achieve sub-second granularity. Currently, SysViz [41] is the only one being able to reconstruct the entire trace of each transaction with sub-second level monitoring. However, SysViz is limited for specific workflow since it relies on the analytical models, and it lacks of scalability because it requires the servers to be connected with the network taps or network switches which support passive network tracing.

milliScope is designed for providing fine-grained monitoring data and linking these data to information about request dependencies and causality so that the researchers would be able to study the milliBottlenecks and the induced very long response time (VLRT) problem easily. The long-tail latency issue is not only the particular concern for mission-critical web-facing applications [8] [9] [22] [46] [40], it is also the important metric in the evaluation of quality of service provided by computing clouds and data centers [11] [61] [65] [76] [77]. To mitigate the long-tail latency, researchers have proposed several bypass techniques which are effective in the specific applications or domain [22] [43] [59].

The potential root causes of the very long response time requests for the web application can occur for very different reasons, such as CPU dynamic voltage and frequency scaling (DVFS) control at the architectural layer [80], Java garbage collection (GC) at the system software layer [79], virtual machine (VM) consolidation at the VM layer [78], and performance interference of memory thrashing [59]. Moreover, Dean et al. outlined the potential causes for the long-tail latency issue of Google's large scale interactive applications [22]. In addition, workload characteristics (e.g., burstiness and request type mix-ratio) [15] [19] [29] [36] [52] [53] [54] [68] and the soft resource (e.g., threads and database connection) [64] [13] [24] [47] [48] [56] [57] [58] [62] [83] [87] allocation have been discussed as the important source for unpredictable performance. In this thesis, we have provided several illustrative scenarios in which milliScope scales the mountain of data to look for the root cause of observed performance anomalies.

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this thesis, we have presented the first millisecond granularity, software-based resource and event monitor, milliScope, for distributed systems (detail in Chapter 2). milliScope provides a fine-grained monitoring framework for different mScopeMonitors to profile execution performance. This reduces the friction for researchers to perform more robust system debugging. milliScope contains several resource mScopeMonitors to monitor system resource utilization. Moreover, we developed our own event mScopeMonitors, which incur negligible overhead because of their integration with the existing logging infrastructure. We present two illustrative scenarios in which the abnormal phenomena look similar at first glance, e.g., one to two orders of magnitude increase in response time over a short period, but they are caused by different system operations: IO activities and dirty page recycling (detail in Chapter 3). Moreover, we validate the accuracy and lightweight characteristics of the event mScopeMonitors and demonstrate the availability and flexibility of millisScope through several evaluations (detail in Chapter 4). With its good performance (low overhead at high frequency) and high accuracy, milliScope is an important contribution towards fine-grained performance debugging of complex n-tier applications in the cloud environments.

6.1 Future Work

Due to the potential depth of the proposed research, the proposed dissertation—even though self-contained and highly significant—can merely be regarded as an initial step towards one of the important goals in the fine-grained system performance debugging: a 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. One topic of particular interest is how the researchers and the system administrator can identify the milliBottlenecks systematically through voluminous monitoring records. While the milliBottlenecks in n-tier systems might exhibit the similar pattern, detecting and diagnosing each individual pathology requires a robust data collection and analytical platform. The seemingly unbounded number of possible ways this pattern could be manifested on top of the complexity of isolating the offending resource is not possible to handle manually. We expect further research could lead us to the holy grail of the automation of milliBottlenecks detections for applications running in distributed cloud environments.

REFERENCES

[1] "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/.`

[2] "Cudera htrace." `http://http://github.com/cloudera/htrace`.

[3] "millscope web page." `http://www.cc.gatech.edu/systems/projects/Elba/project.html`.

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

[5] "Twitter zipkin." `http://http://twitter.github.io/zipkin/`.

[6] Abd-El-Malek, M., II, W. V. C., Cranor, C., Ganger, G. R., Hendricks, J., Klosterman, A. J., Mesnier, M. P., Prasad, M., Salmon, B., Sambasivan, R. R., Sinnamohideen, S., Strunk, J. D., Thereska, E., Wachs, M., and Wylie, J. J., "Ursa minor: Versatile cluster-based storage," in *Proceedings of the FAST '05 Conference on File and Storage Technologies, December 13-16, 2005, San Francisco, California, USA*, 2005.

[7] Adelberg, B. and Denny, M., "Nodose version 2.0," in *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*

[8] Alizadeh, M., Greenberg, A. G., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M., "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, pp. 63–74, 2010.

[9] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M., "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pp. 253–266, 2012.

[10] Arasu, A. and Garcia-Molina, H., "Extracting structured data from web pages," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003.*

[11] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

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

[13] Beltran, V., Torres, J., and Ayguadé, E., "Understanding tuning complexity in multithreaded and hybrid web servers," in *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*, pp. 1–12, 2008.

[14] Bienvenu, M., ten Cate, B., Lutz, C., and Wolter, F., "Ontology-based data access: a study through disjunctive datalog, csp, and MMSNP," in *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*.

[15] Bodík, P., Fox, A., Franklin, M. J., Jordan, M. I., and Patterson, D. A., "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pp. 241–252, 2010.

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

[17] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E., "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, 2008.

[18] Chen, M. Y., Accardi, A., Kiciman, E., Patterson, D. A., Fox, A., and Brewer, E. A., "Path-based failure and evolution management," in *1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California, USA, Proceedings*, pp. 309–322, 2004.

[19] Choi, K., Soma, R., and Pedram, M., "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 18–28, 2005.

[20] Chow, M., Meisner, D., Flinn, J., Peek, D., and Wenisch, T. F., "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.

[21] Codd, E. F., "Relational database: A practical foundation for productivity," *Commun. ACM*, vol. 25, no. 2, pp. 109–117, 1982.

[22] Dean, J. and Barroso, L. A., "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[23] DeWitt, D. J. and Gray, J., "Parallel database systems: The future of high performance database systems," *Commun. ACM*, vol. 35, no. 6, pp. 85–98, 1992.

[24] Diao, Y., Gandhi, N., Hellerstein, J. L., Parekh, S. S., and Tilbury, D. M., "Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Management Solutions for the New Communications World, 8th IEEE/IFIP Network Operations and Management Symposium, NOMS 2002, Florence, Italy, April 15-19, 2002. Proceedings*, pp. 219–234, 2002.

[25] Florescu, D. and Kossmann, D., "Rethinking cost and performance of database systems," *SIGMOD Record*, vol. 38, no. 1, pp. 43–48, 2009.

[26] Fonseca, R., Dutta, P., Levis, P., and Stoica, I., "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.

[27] Fonseca, R., Freedman, M. J., and Porter, G., "Experiences with tracing causality in networked services," in *2010 Internet Network Management Workshop / Workshop on Research on Enterprise Networking, San Jose, CA, USA, April, 2010*, 2010.

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

[29] Freeh, V. W., Lowenthal, D. K., Pan, F., Kappiah, N., Springer, R., Rountree, B., and Femal, M. E., "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 835–848, 2007.

[30] Garcia-Molina, H. and Salem, K., "Main memory database systems: An overview," *IEEE Trans. Knowl. Data Eng.*, vol. 4, no. 6, pp. 509–516, 1992.

[31] Ghemawat, S., Gobioff, H., and Leung, S., "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, pp. 29–43, 2003.

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

[33] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A., "Difference engine: harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, 2010.

[34] Gupta, R. and Haritsa, J., "Commit processing in distributed real-time database systems," in *Proceedings of 17th IEEE Real-Time Systems Symposium*, 1996.

[35] Hellerstein, J. L., Maccabee, M. M., III, W. N. M., and Turek, J., "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.

[36] Isci, C., Contreras, G., and Martonosi, M., "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39 2006), 9-13 December 2006, Orlando, Florida, USA*, pp. 359–370, 2006.

[37] Jeon, M., He, Y., Kim, H., Elnikety, S., Rixner, S., and Cox, A. L., "TPC: target-driven parallelism combining prediction and correction to reduce tail latency in interactive services," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Atlanta, GA, USA, April 2-6, 2016*, pp. 129–141, 2016.

[38] Kapoor, R., Porter, G., Tewari, M., Voelker, G. M., and Vahdat, A., "Chronos: predictable low latency for data center applications," in *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012*, p. 9, 2012.

[39] Kavulya, S., Daniels, S., Joshi, K. R., Hiltunen, M. A., Gandhi, R., and Narasimhan, P., "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.

[40] Kohavi, R. and Longbotham, R., "Online experiments: Lessons learned," *IEEE Computer*, vol. 40, no. 9, pp. 103–105, 2007.

[41] Laboratories, F., "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`.

[42] Lai, C.-A., Kimball, J., Zhu, T., Wang, Q., and Pu, C., "milliscope: a fine-grained monitoring framework for performance debugging of n-tier web services," in *37th International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5 - June 8, 2017*.

[43] Lai, C., Wang, Q., Kimball, J., Li, J., Park, J., and Pu, C., "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*.

[44] Lee, S.-W., Moon, B., Park, C., Kim, J.-M., and Kim, S.-W., "A case for flash memory ssd in enterprise database applications," in *SIGMOD Conference*, pp. 1075–1086, 2008.

[45] Leverich, J. and Kozyrakis, C., "Reconciling high server utilization and sub-millisecond quality-of-service," in *Ninth Eurosys Conference 2014, EuroSys 2014, Amsterdam, The Netherlands, April 13-16, 2014*, pp. 4:1–4:14, 2014.

[46] Li, J., Sharma, N. K., Ports, D. R. K., and Gribble, S. D., "Tales of the tail: Hardware, os, and application-level sources of tail latency," in *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 03 - 05, 2014*, pp. 9:1–9:14, 2014.

[47] Li, Y., Tian, Q., Gao, Q., and Xu, J., "Directional two-dimensional neighborhood preserving projection for face recognition," in *International Conference on Computational Aspects of Social Networks, CASoN 2010, Taiyuan, China, 26-28 September 2010*, pp. 357–360, 2010.

[48] Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J. L., and Parekh, S. S., "Online response time optimization of apache web server," in *Quality of Service - IWQoS 2003, 11th International Workshop, Berkeley, CA, USA, June 2-4, 2003, Proceedings*, pp. 461–478, 2003.

[49] Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., and Kozyrakis, C., "Improving resource efficiency at scale with heracles," *ACM Trans. Comput. Syst.*, vol. 34, no. 2, pp. 6:1–6:33, 2016.

[50] Lowell, D. E. and Chen, P. M., "Free transactions with rio vista," in *SOSP*, pp. 92–101, 1997.

[51] Mace, J., Roelke, R., and Fonseca, R., "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.

[52] Merkel, A., Stoess, J., and Bellosa, F., "Resource-conscious scheduling for energy efficiency on multicore processors," in *European Conference on Computer Systems, Proceedings of the 5th European conference on Computer systems, EuroSys 2010, Paris, France, April 13-16, 2010*, pp. 153–166, 2010.

[53] Mi, N., Casale, G., Cherkasova, L., and Smirni, E., "Injecting realistic burstiness to a traditional client-server benchmark," in *Proceedings of the 6th International Conference on Autonomic Computing, ICAC 2009, June 15-19, 2009, Barcelona, Spain*, pp. 149–158, 2009.

[54] Miftakhutdinov, R., Ebrahimi, E., and Patt, Y. N., "Predicting performance impact of DVFS for realistic memory systems," in *45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Vancouver, BC, Canada, December 1-5, 2012*, pp. 155–165, 2012.

[55] Mogul, J. C., "Emergent (mis)behavior vs. complex software systems," in *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006*, pp. 293–304, 2006.

[56] Olshefski, D. P. and Nieh, J., "Understanding the management of client perceived response time," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/Performance 2006, Saint Malo, France, June 26-30, 2006*, pp. 240–251, 2006.

[57] Osogami, T. and Kato, S., "Optimizing system configurations quickly by guessing at the performance," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2007, San Diego, California, USA, June 12-16, 2007*, pp. 145–156, 2007.

[58] Pariag, D., Brecht, T., Harji, A. S., Buhr, P. A., Shukla, A., and Cheriton, D. R., "Comparing the performance of web server architectures," in *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pp. 231–243, 2007.

[59] Park, J., Wang, Q., Li, J., Lai, C.-A., Zhu, T., and Pu, C., "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.*

[60] Park, J., Wang, Q., Li, J., Lai, C., Zhu, T., and Pu, C., "Performance interference of memory thrashing in virtualized cloud environments: A study of consolidated n-tier applications," in *9th IEEE International Conference on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2, 2016*, pp. 276–283, 2016.

[61] Patterson, D. A., "Latency lags bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, 2004.

[62] Raghavachari, M., Reimer, D., and Johnson, R. D., "The deployer's problem: Configuring application servers for performance and reliability," in *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, pp. 484–489, 2003.

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

[64] Risse, T., Aberer, K., Wombacher, A., Surridge, M., and Taylor, S., "Configuration of distributed message converter systems," *Perform. Eval.*, vol. 58, no. 1, pp. 43–80, 2004.

[65] Rumble, S. M., Ongaro, D., Stutsman, R., Rosenblum, M., and Ousterhout, J. K., "It's time for low latency," in *13th Workshop on Hot Topics in Operating Systems, HotOS XIII, Napa, California, USA, May 9-11, 2011*, 2011.

[66] Sambasivan, R. R., Zheng, A. X., Rosa, M. D., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., and Ganger, G. R., "Diagnosing performance changes by comparing request flows," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA*, 2011.

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

[68] Snowdon, D. C., Sueur, E. L., Petters, S. M., and Heiser, G., "Koala: a platform for os-level power management," in *Proceedings of the 2009 EuroSys Conference, Nuremberg, Germany, April 1-3, 2009*, pp. 289–302, 2009.

[69] Suresh, P. L., Canini, M., Schmid, S., and Feldmann, A., "C3: cutting tail latency in cloud data stores via adaptive replica selection," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pp. 513–527, 2015.

[70] Tak, B., Tang, C., Zhang, C., Govindan, S., Urgaonkar, B., and Chang, R. N., "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.

[71] Thereska, E., Salmon, B., Strunk, J. D., Wachs, M., Abd-El-Malek, M., López, J., and Ganger, G. R., "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.

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

[73] von Behren, J. R., Condit, J., and Brewer, E. A., "Why events are a bad idea (for high-concurrency servers)," in *Proceedings of HotOS'03: 9th Workshop on Hot Topics in Operating Systems, May 18-21, 2003, Lihue (Kauai), Hawaii, USA*, pp. 19–24, 2003.

[74] Waldspurger, C. A., "Memory resource management in vmware esx server," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 181–194, Dec. 2002.

[75] Wang, Q., *A study of transient bottlenecks: understanding and reducing latency long-tail problem in n-tier web applications.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2015.

[76] Wang, Q., Kanemasa, Y., Kawaba, M., and Pu, C., "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.

[77] Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Shimizu, T., Matsubara, M., Kawaba, M., and Pu, C., "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.

[78] Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Shimizu, T., Matsubara, M., Kawaba, M., and Pu, C., "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.

[79] Wang, Q., Kanemasa, Y., Li, J., Lai, C., Cho, C., Nomura, Y., and Pu, C., "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.

[80] Wang, Q., Kanemasa, Y., Li, J., Lai, C., Matsubara, M., and Pu, C., "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.*

[81] Wang, Q., Lai, C.-A., Kanemasa, Y., Zhang, S., and Pu, C., "A study of long-tail latency in n-tier systems: Rpc vs. asynchronous invocations," in *37th International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5 - June 8, 2017.*

[82] Wang, Q., Malkowski, S., Jayasinghe, D., Xiong, P., Pu, C., Kanemasa, Y., Kawaba, M., and Harada, L., "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*.

[83] Welsh, M., Culler, D. E., and Brewer, E. A., "SEDA: an architecture for well-conditioned, scalable internet services," in *Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001, Chateau Lake Louise, Banff, Alberta, Canada, October 21-24, 2001*, pp. 230–243, 2001.

[84] Xu, W., Huang, L., Fox, A., Patterson, D. A., and Jordan, M. I., "Detecting large-scale system problems by mining console logs," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*.

[85] Xu, Y., Musgrave, Z., Noble, B., and Bailey, M., "Bobtail: Avoiding long tails in the cloud," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, pp. 329–341, 2013.

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

[87] Zheng, W., Bianchini, R., and Nguyen, T. D., "Automatic configuration of internet services," in *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pp. 219–229, 2007.

[88] Zhu, T., Li, J., Kimball, J., Park, J., Lai, C.-A., Pu, C., and Wang, Q., "Limitations of load balancing mechanisms for n-tier systems in the presence of millibottlenecks," in *37th International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5 - June 8, 2017*.