Columbo: Low Level End-to-End System Traces through Modular Full-System Simulation

Vision Paper (7 pages total)

Jakob Görgen
Max Planck Institute for Software Systems
Saarbrücken, Germany
jgoergen@mpi-sws.org

Vaastav Anand
Max Planck Institute for Software Systems
Saarbrücken, Germany
vaastav@mpi-sws.org

Hejing Li
Max Planck Institute for Software Systems
Saarbrücken, Germany
hejingli@mpi-sws.org

Jialin Li
National University of Singapore
Singapore
lijl@comp.nus.edu.sg

Antoine Kaufmann
Max Planck Institute for Software Systems
Saarbrücken, Germany
antoinek@mpi-sws.org

ABSTRACT

Fully understanding performance is a growing challenge when building next-generation cloud systems. Often these systems build on next-generation hardware, and evaluation in realistic physical testbeds is out of reach. Even when physical testbeds are available, visibility into essential system aspects is a challenge in modern systems where system performance depends on often sub-*μs* interactions between HW and SW components. Existing tools such as performance counters, logging, and distributed tracing provide aggregate or sampled information, but remain insufficient for understanding individual requests in-depth.

In this paper, we explore a fundamentally different approach to enable in-depth understanding of cloud system behavior at the software and hardware level, with (almost) arbitrarily fine-grained visibility. Our proposal is to run cloud systems in detailed full-system simulations, configure the simulators to collect detailed events without affecting the system, and finally assemble these events into end-to-end system traces that can be analyzed by existing distributed tracing tools.

1 INTRODUCTION

Understanding performance of a modern cloud or datacenter system in depth during development is increasingly difficult. Modern systems are heterogeneous and require more complex and fine-grained interaction between various software and hardware components. Heterogeneous systems with specialized hardware and software components [4, 9, 10, 16, 19, 27–29] improves performance and efficiency, but also at the expense of understandability.

With specialized hardware, such as computational accelerators, SmartNICs, smart SSDs, disaggregated memory, or in-memory compute, overall system performance often critically depends on frequent, *μs*- or *ns*-scale, and asynchronous interactions between hardware and software. To make matters worse, hardware and software components are typically tightly integrated, e.g., through kernel-bypass, resulting in complex request control flows and no common points for easy observability, such as the kernel syscall interface. Attempts to instrument these systems for visibility into performance either provide visibility limited to aggregated data, such as performance counters, or incur prohibitive overheads that also significantly change system behavior.

Building next-generation research systems frequently brings additional challenges with hardware, often because appropriate physical testbeds are out of reach. Some work may explore emerging hardware or hardware features, where hardware may be not yet available or may have substantial bugs or limitations. For work proposing new hardware changes, evaluation is even more challenging, as prototyping hardware takes significantly longer and is often prohibitively expensive. As a result, a thorough end-to-end evaluation for in-depth performance is either not possible, or heavily limited in insights.

Both of these problems — in-depth visibility and lack of physical testbeds — can be simultaneously addressed by simulation. Simulations offer four key advantages: (i) they can provide virtual testbeds for systems where a physical testbed is unavailable. (ii) virtually *unlimited visibility into behavior of the simulated system without affecting system behavior* through detailed simulator logs. Detailed simulator logs, e.g., logging each instruction or cache line accessed by a processor, or reception and transmission time of each packet, might substantially slow down the simulator itself, but have no impact on the simulated system in any way. (iii) they can provide ground-truth information hard to obtain in physical systems, such as precise timing for events or access low-level hardware interactions such as memory bus transactions. For instance, simulation provides a true and precise global clock for all events. (iv) they can get data from multiple components within the same execution context to establish causal links between events originating from different components.

As no individual simulator supports simulating all components needed for most data center systems, modular full system simulation [15, 24], i.e., combining and connecting multiple existing simulators for different components into complete simulated testbeds, is the only available means for simulating complete cloud and datacenter systems. Full-system simulation includes all hardware and software, including applications, libraries, and operating system.

Can existing modular simulators readily address the visibility challenge? Unfortunately, the answer is no. Doing so requires solutions to the following hard issues. First, simulators generate large amount of logging data. As an example, a gem5 simulator [8] instance can generate 100s of GBs log for a few seconds of simulated execution. Second, data from multiple simulators can be difficult to correlate. Establishing causality of events is thus a non-trivial task. Third, log data does not have a common format. They can be semi-structured, and lacks any standardization across simulator types. Lastly, there is a dearth of tooling available to analyze the data generated by such simulations.

To address this, in this paper, we propose combining end-to-end simulations with common monitoring techniques such as distributed tracing [6, 11, 25] represents a means for developers to gain visibility and insights into heterogeneous systems. We have developed Columbo, a prototype efficiently processing the logs generated from simulations to generate traces that can be digested and analyzed existing distributed tracing infrastructure. We show Columbo's efficacy in analyzing heterogeneous systems by using Columbo to show how a user can gain deep insight into issues arising from external factors when using NTP for doing clock synchronization. We believe that the approach provided by Columbo to gain insight into heterogeneous systems is effective and time-saving for developers building heterogeneous distributed systems for the cloud.

2 BACKGROUND AND MOTIVATION

2.1 Modular e2e Simulations

Modular end-to-end simulators are particularly useful for heterogeneous computer systems. Although many simulators have been long established and used in research on various layers of modern systems, these simulators focus on specific components of the entire system. Thus, none of these simulators in isolation enables a true end-to-end evaluation of a complete modern data-center system. For instance, computer architects use gem5 [7] to explore new processors and memory modules etc., network researchers use ns3 [23] to evaluate network protocols and topologies, and hardware researchers use simulators like verilator [26] for hardware RTL design.

Modular end-to-end simulation [15, 18, 24], on the other hand, allows users to combine and connect multiple component simulators to construct a full system simulation. SimBricks [15], for example, provides fixed component simulator interfaces to connect component simulators running as separate processes.

While each component simulator can provide highly detailed information about the behavior of that component without affecting system behavior, manually analyzing and correlating these multiple logs and reasoning about the behavior of the full system is both challenging and extremely time-consuming. Understanding behavior across components also requires a deep understanding of the components and their interactions in the system. The log messages and actions triggered by each component simulator in response to a particular event of interest lack the context of the event. Consequently, users must manually correlate cross-component events using detailed information such as timestamps, DMA request IDs, and accessed memory addresses.

Columbo aims to address these challenges by automatically analyzing the correlations between events generated by different components and visualizing them as distributed traces. By providing a comprehensive overview of the system's behavior, Columbo facilitates easier interpretation and analysis of complex interactions within the system, significantly reducing the effort to understand and debug the system performance.

2.2 Distributed Tracing

Distributed tracing tools are widely used both in open-source [2, 3, 20] and major internet companies [12, 21, 25] to chronicle the structural execution of an end-to-end request [6, 11]. Traditionally, the request's execution flow is captured across different components of a distributed system but the capture is limited to the application layer at each component.

The big advantage of using distributed tracing is that it is particularly useful for troubleshooting cross-component issues as it can provide insight for a failure from different components in a single context using context propagation [17]. A drawback of using distributed tracing is that it adds an extra penalty on the performance of the application being traced leading practitioners to use sampling techniques to effectively limit the data being traced [13, 14, 25] which can cause a potential loss of edge-case data [30]. Moreover,the source data for distributed traces is currently limited to the data generated by the applications and no data from lower layers in the system software stack.

2.3 The Case for Combining Distributed Tracing with E2E Simulation

Using distributed tracing with simulations provides a unique opportunity to leverage the full power of distributed tracing without worrying about the potential drawbacks of using distributed tracing. As end-to-end simulation is not a performance or a time-critical task, we can run distributed tracing at full tilt to generate traces for all execution flows into the system without requiring any kind of sampling to artificially limit the number of traces. Moreover, we can enrich distributed traces with fine-grained information produced from simulators to capture detailed execution flows from different hardware components to generate hardware-enriched traces to provide higher visibility into heterogeneous systems.

3 DESIGN

3.1 Design Goals

To accurately analyze the performance of a distributed system built on a heterogeneous computing environment, we have the following design goals for Columbo.

- End-to-end visibility: Provide the full system information including host, NIC, network with the entire software stack from application to operating system.
- Extensibility: Allow easy integration and support of new component simulators into the tracing system.
- Arbitrarily detailed information: Flexibly control the level of detail for logging based on the need.
- Transparent: Collect complete system log for debug and analyze without affecting distributed system performance

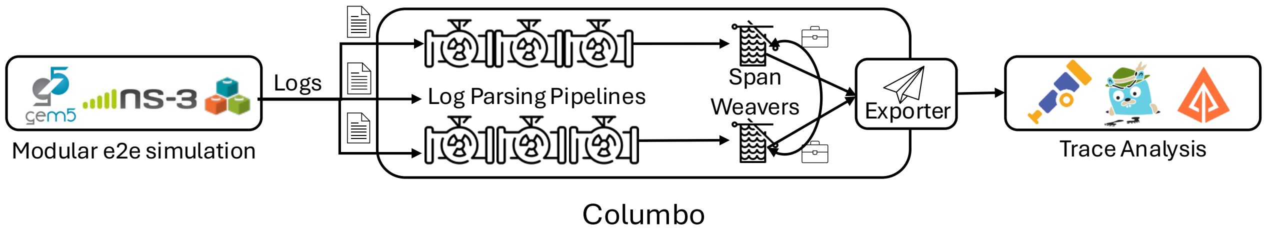

Figure 1: Columbo overview

3.2 Challenges

While combining modular end-to-end simulation with distributed tracing provides a unique opportunity to gain deep insight into the system, we need to overcome many challenges to efficiently process the generated log files to produce information-rich traces that can produce actionable insight.

Non standardized data formats. We are faced with the issue of non standardized log file output formats when running modular full system simulations. Standalone simulators for different components have different log file formats and their respective log files contain different information. Moreover, different simulators of the same kind log information in different formats. There is a significant lack of standardization of simulator log data which makes it difficult to easily manage and connect the simulation data for gaining insight.

Large amounts of data. Simulators can provide arbitrary deep visibility into what a simulated system does. Naturally, this visibility come at the price of huge amounts of log file data being generated. Thus, we must be able to efficiently handle vasts amount of data coming from multiple simulators. Moreover, a user might only be interested in a small subsection of the data for their investigation. Given the large amount of data generated, the user might not be able to easily filter out and find the relevant information.

Data correlation across simulator boundaries. As we use log files from modular, independent, unmodified simulators, making causal connections is difficult across simulator boundaries is non-trivial for two reasons. First, we cannot make any assumptions about the topology being simulated. Thus connecting events across simulator boundaries requires domain-specific knowledge of the topology as well as the simulation to be able to explicitly connect the data. Second, we cannot use any explicit context propagation techniques as we do not want to modify simulators because modifying simulators will break the modularity as well as make the system less extensible to new simulators and components.

Trace analysis. Users of Columbo need an easy way to analyze the traces created. Analysis tools must support efficient querying, expressive analysis, as well as a usable interface for users to interact with the generated traces.

3.3 Overview

We have designed Columbo to be able to process logs from modular end-to-end simulations to generate distributed traces. These traces can then be exported existing trace analysis tools for further processing and utilization by developers. Figure 1 shows the current design architecture of Columbo. Simulation logs are input to the Columbo processing pipeline. To convert these logs into traces, Columbo uses simulator-specific parsers to generate *type-specific event streams* to generate a standard set of events for each simulator type. The generated event streams are further processed by simulator-specific *pipelines* where events can be filtered, modified, or removed depending on the requirements of the user. Finally, the filtered events are passed to the *SpanWeavers* which coalesce the events into spans. *SpanWeavers* also do context propagation to correctly connect spans from different simulators.

3.4 Type-Specific Event Streams

We use modular full-system simulation consisting of multiple simulators, each of which simulates a specific component. Each simulator creates logs in an ad-hoc custom format with no standardization. This leads to two different standardization challenges.

First, simulators of different types have different types of events. For example, consider the events generated by a NIC simulator and host simulator during the simulation of a packet transmission. The host simulator might send a packet during a simulation. For the host simulator, the generated events include function call events to invoke the NIC's driver as well as a mmio write event responsible for instructing the attached NIC to send a packet. The events generated on the NIC side are the dma access events generated by the NIC to read the packet data from the host and a packet transmission event when the NIC finally puts the packet onto the wire to send it.

Second, there is a lack of standardization of log formats across different simulators for the same type. For example, the logfile format of the gem5 host simulator [7] is different from the logfile format of the QEMU host simulator [22].

To fix these issues, we introduce type-specific event streams. We create standard types of events a simulator can emit for every given simulator type. This handles data from two different simulators of the same type in a standardized way as each simulator of a specific type strictly adheres to the same set of events. This eliminates the differences between events generated from different simulator types and enables supporting new simulators with little effort as it only needs to provide a parser for the simulator's specific logfile format has to be provided.

3.5 Pipelines

Depending on the granularity at which a given simulator operates, the logfiles generated by the simulator can contain large amounts of data. For example, logfiles generated by gem5 [7] can have hundreds

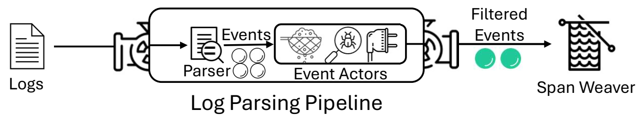

Figure 2: Columbo pipelines overview

of Gbs or even TBs in size. Additionally, not all the data generated by a simulator might be useful for the user.

To solve the problem of dealing with the large amounts of data generated, we introduce the notion of simulator-specific pipelines. Simulator-specific pipelines are used by Columbo to stream and process the data and events contained in the logfiles generated by the simulators. Figure 2 shows an example of how multiple simulator-specific pipelines operate in conjunction together.

A pipeline consists of a producer, multiple optional actors, and a consumer. Producers read and parse the log file of a simulator and generate an event stream as described in §3.4. Actors serve as additional operators on the event stream within a pipeline, for example, to filter events from an event stream or to modify the event stream to resolve a function's address to the function's name. Consumers are the ending stage of a simulator specific pipeline and are called *SpanWeavers*. SpanWeaver group the events of an event stream into traditional distributed tracing spans and are responsible for trace context propagation, thus creating causal relationships between spans. An example of a simulator specific pipeline is depicted in Figure 2. The various components of a pipeline (i.e. producer, actors & consumer) communicate with each other via message queues that may be distributed over the network.

3.6 Context Propagation

As we use unmodified simulators for the simulation, using explicit mechanisms to propagate trace context is infeasible as it would require non-trivial modifications to the simulators. Columbo relies on a form of implicit context propagation in which the logic of when to propagate context is baked into the SpanWeavers operating on a type-specific event stream. Therefore, a span weaver instance is only responsible for the creation of spans and the propagation of trace context in regards to the event stream it operates on.

The choice of when to propagate the context is provided to the user as the user is responsible for configuring events from the event stream into spans. This is because the topology of the system being simulated is different for each experiment, so Columbo cannot hardcode any of the context propagation a priori. Columbo's current prototype uses a simple form of context propagation in which we distinguish two cases.

Intra SpanWeaver propagation. SpanWeavers create spans from the events contained in the event stream it operates on. For a specific simulator's event stream, a SpanWeaver creates a variety of spans, each reflecting a unit of work done in the respective simulator. For example, for the event streams of host simulators, the SpanWeaver might create many different spans such as a span representing a syscall, a span representing a mmio-write access to an attached device, among others. Spans created within the same SpanWeaver can have a causal relationship. For example, a syscall might trigger the host to write a register of an attached device. Therefore, context must be propagated between two such spans. The logic to propagate trace context between spans within a SpanWeaver must be generically programmed into a SpanWeaver and requires domain specific knowledge from the user.

Inter SpanWeaver propagation. Similar to the causal relationships between spans within a SpanWeaver instance, spans created by different SpanWeaver instances may also have causal relationship. For example, consider a span representing a mmio-write access to a register of an to the host attached device. This mmio-write will cause events in the simulator simulating the attached device which in turn will be grouped into a logical unit of work i.e. a span by a SpanWeaver. Between the two spans, created by different SpanWeaver instances, a causal connection must be made to preserve the relationships between different events. This kind of context propagation becomes necessary when simulators communicate through natural boundaries that also exist in real system like PCIe or Ethernet. Therefore, programmers must take into account when events cause events in attached simulators, i.e., a component would communicate over PCIe or Ethernet in a real system, and propagate context to another SpanWeaver instance in such a case. This is done through message passing queues that SpanWeaver instances share. In situations where a span might cause actions in a different simulator, the SpanWeaver must push context to that queue for the SpanWeaver of the other simulator. Vice versa, in case a span might have been caused by an attached simulator a SpanWeaver must poll context from that queue in order to make the causal connection.

To make it easy for users to do context propagation, we limit the number of different types of spans that can be generated from an event span. This allows users to programmatically encode the causal relationships between spans.

3.7 Exporter

Once a SpanWeaver has finished a span and made all the necessary causal connections, it passes the span to an exporter. Exporters are used by Columbo to export spans to external tools from the distributed tracing community, such as Jaeger [2] or OpenZipkin [3]. These tools provide Columbo users with a graphical representation of the created spans and traces. Since such tools may use a different internal representation of spans and traces than Columbo, the exporters are also responsible for converting Columbo's internal span representation into the representation required by the tool a user wants to export traces to, before actually exporting/sending traces to the respective tool.

3.8 Online Analysis

It is desirable to run Columbo in parallel to the actual full-system simulation without the need to persist the log files written by the simulators for analysis after the simulation finished. This is necessary as users might not have hundreds of GBs of disk space at their disposal to persist logfiles for offline analysis.

Therefore, Columbo provides an online mode with the support of Linux named pipes. Each simulator in the simulation uses a Linux named pipe as the destination for it's log events. Columbo is started in parallel to the simulation to read from the named pipes to create traces. This allows Columbo to create traces without requiring any persistence of the log data.

Simulator Type	**# Supported Event Types**	**# Supported Span Types**
Host	16	6
NIC	9	4
Network	3	1

Table 1: Supported simulator types and the number of supported events and span types respectively.

4 IMPLEMENTATION

The current Columbo prototype is completely implemented in C++ and is tailored for use together with SimBricks [15] simulations. It consists of 19,000 lines of code and supports three simulators: the gem5 simulator [7] to simulate end-hosts, the ns3 network simulator [23] as well as the SimBricks Intel i40e NIC behavioral model simulator [15]. The amount of supported event- and span-types for each of these simulators is listed in Table 1.

Columbo scripts. Columbo harnesses logfiles written by simulators during a modular full system simulation. As Columbo is decoupled from the actual simulation and the simulators used, Columbo does not automatically know about the simulated topology and the concrete simulator instances used for the components involved. Therefore, a user has to orchestrate the trace creation by providing a Columbo *Script*. A Columbo *Script* is a small C++ program with the purpose of composing simulator specific pipelines similar to ns3 [23] requiring users to write small C++ programs in order to create experiments. To make it easy for users to write Columbo *Scripts*, Columbo provides predefined building blocks like logfile parsers, event stream actors, SpanWeavers, and predefined components to export the traces created to external tools for visualization and analysis. Using these building blocks users can easily compose simulator specific pipelines in order to created low level end-to-end system traces trough simulations.

5 CASE STUDY: CLOCK SYNCHRONIZATION

In this case study, we consider the scenarios where we want to synchronize the clocks in our system. To do so, we need to synchronize the clock of the client with the clock of the server. The client and server are connected over the network which is also servicing background traffic generated from other systems and applications. Checking if distributed clocks are synchronized is a difficult problem as there is no ground truth available and it is impossible to guarantee that the measurements taken at different clocks will be done so at the exact moment of time.

Instead, to check if the clocks are synchronized, we turn to simulation where we simulate the client, the server, and the underlying network as shown in Figure 3. We simulate the topology with SimBricks [15]. The topology consists of two hosts, the client and server, each connected to a NIC, connected over a network consisting of two switches. To simulate the two hosts, we used the gem5 simulator [7], to simulate the network we used ns3 [23] and to simulate the two NICs we used SimBricks's Intel i40e NIC behavioral model simulator [15].

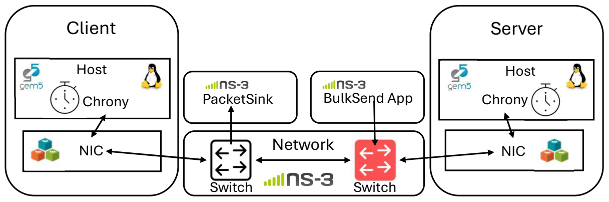

Figure 3: Columbo evaluation topology

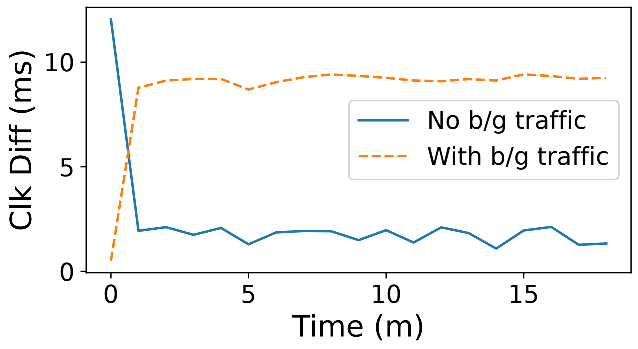

Figure 4: Measured difference between the system clocks of the client and server

To synchronize the clocks we run chrony [1] on both the client and server. We configure chrony to use NTP for doing clock synchronization. We check if the clocks can be synchronized under two different conditions. In *Scenario 1*, we check if the clients can be synchronized when there is no background traffic from other applications. This will help us establish a baseline of how well the clocks can be synchronized in best conditions. In *Scenario 2*, we check if the clients can be synchronized when there is a lot of background traffic from other applications. To simulate the background traffic, we use the BulkSendApplication and PacketSink applications from NS3 to send TCP packets from the application to the sink. We configured the BulkSendApplication to transmit packets at a data rate that exhausts the link between the two switches.

We ran all experiments on a physical host with two Intel(R) Xeon(R) Gold 6336Y CPU @ 2.40GHz, 24 cores each, with 8x 32GiB (256GiB) of memory. During all experiments we enabled log output for the simulators used, i.e. we enabled logging statements when compiling the Intel i40e NIC behavioral model, we used and attached ns3 trace sources, and we used gem5 in the *opt* version together with debugging flags we passed via the command line.

We measure the difference between the client and server system clocks. To ensure that we take a clock reading at the same time, we make use of the global clock available to us in the simulation. Each clock reading has a corresponding timestamp with respect to the global clock. This allows us to apply corrections and ensure that we are reading the clocks at the same time. Figure 4 shows the measured difference in the clocks for both scenarios. We notice that the system does a good job of synchronizing the clocks in the scenario where there is no background traffic. However, in the

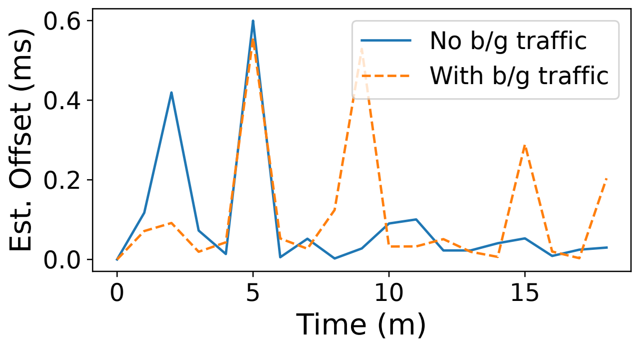

Figure 5: Clock offset between client and server clocks estimated by chrony

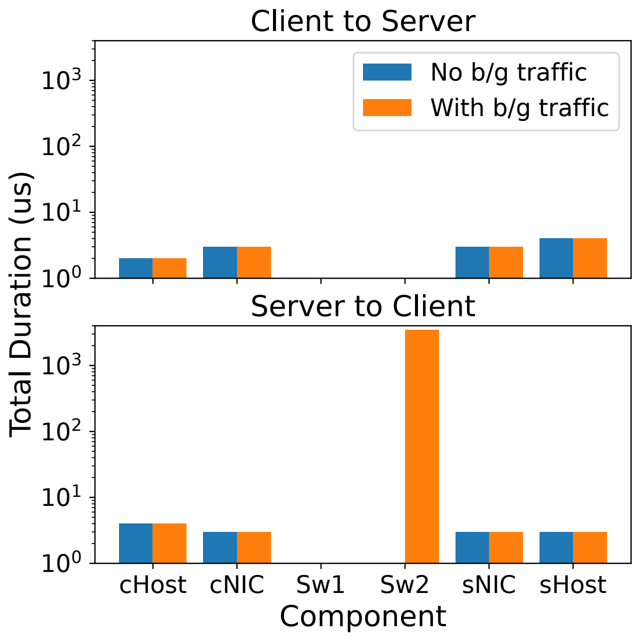

Figure 6: Breakdown of the time spent per component for clock synchronization request

scenario with background traffic, the clock synchronization does not achieve the same accuracy as the baseline.

To figure out why the client and server clocks are not synchronized when there is background traffic, we investigate further and look into the clock offset calculated by chrony. Figure 5 shows the estimated clock offsets for both the scenarios. For both scenarios, the clock offsets have almost identical trends but we know that this is not accurate as our system clock measurement is still not accurate. Thus, we can conclude that there is something odd happening during our clock synchronization but at the moment we do not have enough information to figure out the odd behavior.

To figure out the odd behavior, we need to dive deeper into the clock synchronization process between the server and client. Figure 6 shows the total time spent per component for the two different scenarios. We break it down into two different parts - (i) the synchronization request sent from the client to the server, (ii) the synchronization response sent from the server to the client. As we can see that the big difference in the two situations is that the response spends considerably large amount of time in the switch that is connected directly to the BulkSendApplication. Thus, as the request and response messages take different durations, this makes NTP inaccurate as one of the requirements for NTP is that both the request and response should take a similar amount of time. The added visibility allows us to easily figure out the root cause of the inaccuracy in our synchronization.

6 CHALLENGES & OPPORTUNITIES

Correct Context Propagation: Currently, Columbo infers causal relationships between events without explicit context propagation. This is error-prone in scenarios with multiple parallel events triggered by different components. This error could potentially be prevented by explicit context propagation, but it requires a significant and error prone implementation effort. Moreover, this approach would undermine Columbo goal of allowing for easy integration of new simulators. Thus, correctly doing context propagation in all scenarios remains a significant future challenge.

Gaining insight into deployed systems: Deep insight into deployed systems remain impractical due to overhead. The emergence of heterogeneous hardware systems with novel hardware components further complicates this. As components are developed independently, there is little standardization of the granularity of metrics, logs, or traces available from these components. Operators rely on bespoke solutions for gaining insight into these systems [5].

We believe Columbo provides an opportunity to solve in-production bugs by connecting real executions to simulation. Production systems do not offer the opportunity to gain deep insight into the system as that would drastically impact the performance. However, data collected in production could be used in simulation with Columbo to gain the required visibility into existing systems and potentially solve bugs. We believe that this connection represents an exciting avenue for research in heterogeneous cloud systems.

Tracking down corner-cases: Tracking down rare abnormal behavior in distributed systems is often challenging with existing tracing tools. The root cause is often not solely caused events captured in traces, but also heavily depends on the overall hardware and software state. Columbo builds on typically deterministic simulations, thus users can easily reproduce the problematic behavior of the system with more detailed logging. We expect this significantly enhances debugging efficiency.

7 CONCLUSION

This paper presents a novel approach to gain in-depth understanding of cloud system behavior. We propose leveraging detailed full-system simulations to capture (almost) arbitrarily fine-grained events across software and hardware layers. Columbo seamlessly integrates with any existing simulator generating log files, requiring no changes to the simulators themselves. Since most existing simulators already provide detailed logs, no further instrumentation is needed. Instead, Columbo requires users to create a parser specific to each simulator's log format to translate those logs into a type-specific event stream Columbo can understand. These event streams are then assembled into end-to-end system traces, compatible with existing distributed tracing tools for analysis.

The presented approach offers unique advantages. Firstly, simulations eliminate the performance concerns associated with real-world distributed tracing. Secondly, simulations allow us to enrich these traces with detailed information from different hardware components, creating hardware-enriched traces for unparalleled visibility into heterogeneous systems. By combining the power of distributed tracing with the control offered by simulations, Columbo empowers us to gain a deeper understanding of systems, facilitating efficient performance analysis and optimization.

REFERENCES

[1] chrony project. https://chrony-project.org/, 2024. Retrieved Jul 3, 2024.

[2] Jaeger: open source, distributed tracing platform. https://www.jaegertracing.io/, 2024. Retrieved Jul 3, 2024.

[3] Openzipkin · a distributed tracing system. https://zipkin.io/, 2024. Retrieved Jul 3, 2024.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2016.

[5] Valentin Andrei, Riham Selim, Hao Wang, and Lei Tian. System at scale: Ai observability. Accessed July 2024 from https://atscaleconference.com/systemscale-ai-observability/, 2023.

[6] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, volume 4, pages 18–18, 2004.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, August 2011.

[9] Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, et al. The future of fpga acceleration in datacenters and the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(3):1–42, 2022.

[10] Amin Firoozshahian, Olivia Wu, Joel Coburn, and Roman Levenstein. Mtia v1: Meta's first-generation ai inference accelerator. Accessed July 2024 from https://ai.meta.com/blog/meta-training-inference-accelerator-AI-MTIA/, 2023.

[11] Rodrigo Fonseca, George Porter, Randy H Katz, and Scott Shenker. {X-Trace}: A pervasive network tracing framework. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, 2007.

[12] Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, et al. Canopy: An end-to-end performance tracing and analysis system. In *Proceedings of the 26th symposium on operating systems principles*, pages 34–50, 2017.

[13] Pedro Las-Casas, Jonathan Mace, Dorgival Guedes, and Rodrigo Fonseca. Weighted sampling of execution traces: Capturing more needles and less hay. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 326–332, 2018.

[14] Pedro Las-Casas, Giorgi Papakerashvili, Vaastav Anand, and Jonathan Mace. Sifter: Scalable sampling for distributed traces, without feature engineering. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 312–324, 2019.

[15] Hejing Li, Jialin Li, and Antoine Kaufmann. SimBricks: End-to-end network system evaluation with modular simulation. In *2022 ACM SIGCOMM Conference on Data Communication*, SIGCOMM, 2022.

[16] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 574–587, 2023.

[17] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. Pivot tracing: Dynamic causal monitoring for distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 35(4):1–28, 2018.

[18] Christian Menard, Jeronimo Castrillon, Matthias Jung, and Norbert Wehn. System simulation with gem5 and SystemC: The keystone for full interoperability. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS, 2017.

[19] Oleg Obleukhov and Ahmad Byagowi. How precision time protocol is being deployed at meta. Accessed Jul, 2024 from HowPrecisionTimeProtocolisbeingdeployedatMeta, 2022.

[20] OpenTelemetry. High-quality, ubiquitous, and portable telemetry to enable effective observability. Accessed July 2024 from https://opentelemetry.io/.

[21] Maulik Pandey. Building netflix's distributed tracing infrastructure. Accessed July 2024 from https://netflixtechblog.com/building-netflixs-distributed-tracing-infrastructure-bb856c319304, 2020.

[22] QEMU Authors. QEMU – the FAST! processor emulator. https://www.qemu.org/, 2022. Retrieved Feb 2, 2022.

[23] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[24] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, March 2011.

[25] Benjamin H Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. 2010.

[26] Wilson Snyder. Verilator – the fastest verilog HDL simulator. https://www.veripool.org/wiki/verilator, 2022. Retrieved Feb 2, 2022.

[27] Eran Tal, Nicolaas Viljoen, Joel Coburn, and Roman Levenstein. Our next-generation meta training and inference accelerator. Accessed July 2024 from https://ai.meta.com/blog/next-generation-meta-training-inference-accelerator-AI-MTIA/, 2024.

[28] Maria Xekalaki, Juan Fumero, Athanasios Stratikopoulos, Katerina Doka, Christos Katsakioris, Constantinos Bitsakos, Nectarios Koziris, and Christos Kotselidis. Enabling transparent acceleration of big data frameworks using heterogeneous hardware. *Proceedings of the VLDB Endowment*, 15(13):3869–3882, 2022.

[29] Mingran Yang, Zhizhen Zhong, and Manya Ghobadi. On-fiber photonic computing. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 263–271, 2023.

[30] Lei Zhang, Zhiqiang Xie, Vaastav Anand, Ymir Vigfusson, and Jonathan Mace. The benefit of hindsight: Tracing {Edge-Cases} in distributed systems. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 321–339, 2023.