MicroMon: A Monitoring Framework for Tackling Distributed Heterogeneity

Babar Khalid[*+], Nolan Rudolph[†+], Ramakrishnan Durairajan[†], Sudarsun Kannan[*]
[*]*Rutgers University,* [†]*University of Oregon* ([+]*co-primary authors*)

Abstract

We present MicroMon, a multi-dimensional monitoring framework for geo-distributed applications using heterogeneous hardware. In MicroMon, we introduce micrometrics, which is a set of fine-grained hardware and software metrics required to study the combined impact of heterogeneous resources on application performance. Besides collecting micrometrics, in MicroMon, we propose anomaly reports and concerted effort between the programmable switches and host OSes to reduce the overhead of collecting and disseminating thousands of micrometrics in WAN. We evaluate the MicroMon prototype on Cassandra deployed across multiple data centers and show 10–50% throughput gains in a geo-distributed setting with storage and network heterogeneity.

1 Introduction

Modern enterprises deploy large scale applications across tens of geographically-distributed data centers (DCs) to overcome the storage and wide-area network (WAN) limitations within and across DCs. These DCs use heterogeneous storage (e.g., SSD, NVMe, Harddisk), and WAN resources (e.g., fiber optics, InfiniBand), among others[1], to avoid vendor lockout, achieve scalability, and reduce operational and end-user costs. Consequently, applications are beginning to reap the benefits of heterogeneity; for example, the use of a combination of fast-but-expensive SSDs as well as bandwidth- and latency-constrained HDDs with large capacity [20, 25].

While resource heterogeneity is beneficial, realizing those benefits introduces several application-specific performance (e.g., low latency, high throughput, etc.) and correctness (consistency, durability, etc.) challenges. These challenges stem from the lack of understanding of the combined impact of heterogeneous resources and lightweight monitoring frameworks that can provide end-to-end monitoring of multi-dimensional resources across DCs. In this position paper, we posit that there is a fundamental disconnect between the requirements of geo-distributed applications using heterogeneous resources and today's coarse-grained monitoring frameworks.

First, state-of-the-art monitoring frameworks are unidimensional i.e., they either monitor hosts [1, 21], storage devices [16, 27, 28], or network [17, 29] in isolation. For example, while the innovations in programmable switch dataplanes [7, 18] are compelling, they are siloed and do not work holistically with the OS at end hosts. Second, current frameworks cannot monitor large-scale applications with diverse requirements running on heterogeneous resources. Their monitoring techniques only support coarse-granular monitoring of host and network resources [6, 11, 14, 15], or unaware of host-level and network-level heterogeneity [3]. Take the example of modern NoSQL database such as Cassandra [2] that ship snitching mechanisms [6] to monitor replicas at a coarse-grained scope, collecting information such as round-trip times, access, and wait times, resource utilization and re-routing requests across replicas.

To tackle these challenges, this paper presents **MicroMon**, a monitoring framework that efficiently collects, disseminates, and processes the combined impact of storage and WAN heterogeneity—i.e., heterogeneous hardware and software resources—for improving the performance of Cassandra.

To overcome the problem of coarse-grained monitoring in general, MicroMon introduces *micrometrics*, which is a set of fine-grained hardware and software metrics required to study the combined impact of heterogeneous resources on application performance. In our current prototype, MicroMon collects several fine-grained storage and network hardware micrometrics (e.g., storage SMART counters, network packet drops) and software micrometrics (e.g., host-level page cache, block and network stack's I/O queues length) in addition to straight forward hardware metrics (choice of disks and network bandwidth and latency across replicas).

Next, collecting and disseminating thousands of micrometrics in WANs could impact application performance. MicroMon overcomes this challenge in two steps. First, MicroMon introduces *anomaly reports*, where for all possible host-level micrometrics, the host OS only reports anomalies.

[1]We intend to consider compute (e.g., CPU, GPU, TPU) and memory (e.g., DRAM and NVM) as part of future work.

We note that current OSes could be augmented (in addition to what they provide) to report host-level anomalies easily. In a similar vein, MicroMon works in concert with host OSes and network resources (e.g., switches) and reports aggregated events (similar to anomaly reports) by leveraging advances in programmable switch data planes. This is a drastic shift from network telemetry efforts whose sole focus is on network-specific events (e.g., flooding attack-induced congestion).

As a driving use case, we study and deploy MicroMon's prototypic design on Cassandra for optimal replica selection. Our preliminary evaluation of MicroMon on Cassandra for a geo-distributed deployment[2] on CloudLab shows 10–40% performance gains compared to using Cassandra's widely-used coarse-grained monitoring (Snitch).

2 Background

Geo-distributed applications and their diverse requirements. With increasing data processing, analytics, and storage demands, geo-distributed applications are becoming a lifeline of modern enterprises and content providers, spanning across several DCs. These applications range from compute-intensive streaming (e.g., Apache Spark, Hadoop) and batch processing applications to I/O-intensive data serving applications such as NoSQL Cassandra [2], Google Spanner [9], Amazon's Dynamo that must support millions of operations with microsecond-level latency. In addition, these geo-distributed applications have varying levels of consistency, availability, security, and partition tolerance requirements. For example, compared to streaming applications, data serving applications such as Spanner demand higher availability and stronger consistency; hence data placement and replica selection becomes a key aspect of the application design.

For example, Cassandra allows the end-user request to land on any quorum-based replica node. For data partitioning and request routing across replicas, Cassandra (and other similar applications) use consistent hashing [10]. Each node gets assigned to some key range and acts as a coordinator node responsible for replication. The coordinator is responsible for replication of data on different nodes and replicating to other $n-1$ replica nodes. For replica selection and request routing, Cassandra uses *snitch* [6] in each of its nodes, which informs about the network topology, workload, historical latency conditions, and the detection of failing or slow nodes. Snitch allows Cassandra to distribute replicas according to the replication strategy by grouping machines into datacenters and racks. In this work, we use Cassandra as an example application (specifically, in our evaluations) to demonstrate that it has diverse requirements and that it is unaware of resource heterogeneity.

[2]The aforementioned challenges prevail not only across geo-distributed DCs but also within a single DC deployment of applications such as Cassandra. We note that our MicroMon is applicable for such a specific case, too: that is, instead of considering WAN micrometrics (e.g. delays between router hops), our solution can be easily configured to consider an intra-DC network micrometrics (e.g. delays between core switches and servers).

Programmable switches and network telemetry. Monitoring the state of the network—also known as network telemetry—has been well-studied by the community for decades. Telemetry information is collected at different granularities (e.g., packets, flows, samples of flows, etc. [4,5]) by installing network taps at key locations—along with switches—in the network. Recently, programmable switches [7] are slowly replacing the traditional setup; this is primarily due to their increased flexibility and functionality. The collected telemetry is sent either out-of-band (directly) or in-band (via dataplane packets to a "telemetry sink") to a remote inference engine or a controller and further actions are taken [4, 17, 29]. Unlike traditional network telemetry, programmable switches enable collection and dissemination of processed telemetry reports that are succinct (e.g., reporting heavy hitters vs. sending raw packets or counters to the collector) and that captures the state of the network effectively.

3 Motivation

Our research is motivated by the need to close the semantic gap between (a) the growing requirements of geo-distributed applications and the heterogeneity of the DC resources on which they are deployed as well as (b) the paucity of monitoring frameworks to capture fine-grained telemetry information at multiple dimensions (i.e., at the heterogeneous resource level and at the end-to-end application level).

3.1 Growing Resource Heterogeneity and Application Requirements

While applications are scaling across DCs that are geographically distributed, the hardware resources of DC systems are also becoming more and more heterogeneous [16, 19]. For example, take the case of storage heterogeneity: though DCs are moving towards an era of fast SSDs and nonvolatile memory (NVMe), traditional low cost-but-slower alternatives such as HDDs continue to be a vital part of a storage tier, thereby increasing storage heterogeneity across datacenters [19].

In addition, the heterogeneity of resources impacts application management, request routing, data placement, replica selection, and has a direct impact on applications' performance and requirements. For example, Cassandra is highly network-intensive as well as storage-intensive. Unfortunately, such an application must quickly decide on which replica to route to and what request to perform during data placement or fetching. Unfortunately, today's geo-distributed applications (1) are unaware of resource heterogeneity (e.g. storage SSD vs. hard disk), (2) network dynamism (e.g., routing delays, link outages, and path failures) [26], and (3) fine-grained host-level hardware metrics (e.g., the storage hardware's program-erase (P/E) cycles, temperature, I/O traffic), or software bottlenecks (e.g., application page cache state, storage I/O queue, TCP queue occupancy, segmentation Qdisc queue). We refer to these fine-grained metrics as micrometrics.

3.2 Lack of Multi-dimensional Monitoring

The problem is complicated further by the lack of multi-dimensional resource monitoring frameworks. For example, deploying Cassandra in a WAN is fraught with challenges including WAN heterogeneity, path failures, unplanned outages, and performance and topological changes. Prior efforts attempt to address these challenges—*in isolation*—by creating topology awareness [14], latency and bandwidth awareness [15], network-level multi-dimensional resource monitoring and aggregation [11], and snitching mechanisms [6]. While the above-mentioned efforts are as compelling as ever, they are mostly one-dimensional. Other multi-dimensional work such as [13] deal with CPU and bandwidth heterogeneity for long-running stream processing systems and is not designed in the context of geo-distributed applications in general. In addition, these approaches are an ill-fit for latency-sensitive applications. Moreover, these approaches are mostly focused on application-level resource adaptability and a lack of high-resolution resource monitoring does not have a significant impact on performance.

In the networking front, while the programmable switches and network telemetry efforts provide the needed micrometrics (e.g., path that a packet takes, number of unique flows per second, heavy hitters), they are network specific, and we require concerted effort between the programmable switches and host OSes.

4 Design of MicroMon

4.1 Micrometrics Selection

A key technical challenge towards the design of MicroMon lies in identifying and selecting the key software- and hardware-based micrometrics to collect from all the resources (e.g., network, storage).

4.1.1 Mapping Micrometrics to Resource Sensitivity

One problem is that the choice of micrometrics could substantially vary across applications, applications perceived performance-metrics, and DC/cloud deployments. For example, latency sensitive microservices, and geo-distributed key-value stores like Cassandra and memcached [24] are highly latency sensitive and are directly dependent on I/O latency with short request times compared to previously studied geo-analytics such as Apache Spark that use compute-intensive and throughput sensitive queues. We observe that one way to scope the problem of micrometrics selection is by mapping applications to hardware and software resource sensitivity.

4.1.2 Storage H/W and S/W Micrometrics

To scope the solution in our current design of MicroMon, we focus on I/O-intensive Cassandra in this paper. Without loss of generality, these metrics are applicable to several large classes of I/O-intensive applications.

Interplay between hardware micrometrics. Most DCs monitor and collect system (and storage) health and performance metrics. For storage, this includes straightforward metrics such as latency and bandwidth as well as device-level SMART counters such as Raw Read Error Rate (read error rate), Program Fail Count (write error count), device block wear, and temperature. We note that these metrics are used by prior studies in isolation towards data placement and load imbalance [22]. We posit the importance of considering the interplay between such hardware micrometrics, network performance, and application's data access patterns. For example, in NoSQL store replica selection and request routing, a node with low network latency but high program error (PE) count can still be used as a reliable read replica as long as read error rates and temperature do not increase. Unfortunately, *current one-dimensional monitoring systems simply quarantine the entire physical node with high PE, routing requests to nodes with higher network latency* [22].

Interplay between Software micrometrics. Storage software micrometrics such as per-node page cache and outstanding block I/O requests could play a crucial role towards request routing and replica selection. For example, in Cassandra's LSM design, the software storage is maintained as String Sorted Tables (SST tables) composed of several files, with each file storing a range of key-value pairs. As we will discuss in our evaluation, accessing data from a replica with hard-drives but significantly large page cache state can significantly boost throughput and lower latency compared to accessing data from a SSD replica without pagecache. However, *current monitoring and replica selection mechanisms clearly lack such semantic awareness*.

4.1.3 Network H/W and S/W Micrometrics

While several metrics are available via Simple Network Management Protocol (SNMP) including sensor information (e.g., temperature, fan status, etc.), collections of hardware and software micrometrics from the network *and* joint decision with micrometrics host-specific micrometrics (in § 4.1.2) are critical to the success of our work. To this end, we identify several network hardware-specific micrometrics including status of relevant links (ifOperStatus in SNMP to link speed), SFP status (if available); resource information (e.g., CPU load, memory usage) via SNMP get, up/down status of devices/port, queue and buffer utilization, and link-specific information including Q-drops, SNR; among others. For example, prior effort utilized the signal quality information (i.e., Q-drop) to predict future outages in large optical backbones [12]. Indeed, we propose to leverage these micrometrics and expose them to applications.

Similar to the hardware micrometrics, recent innovations in programmable data planes support seamless extraction of software micrometrics from the network including port violation, QoS statistics including drops per queue, packets per Differentiated Services Code Point (DSCP), packet errors and discards, and aggregated network states including heavy hitters, flow arrival rate, etc. to further enhance the performance

of applications.

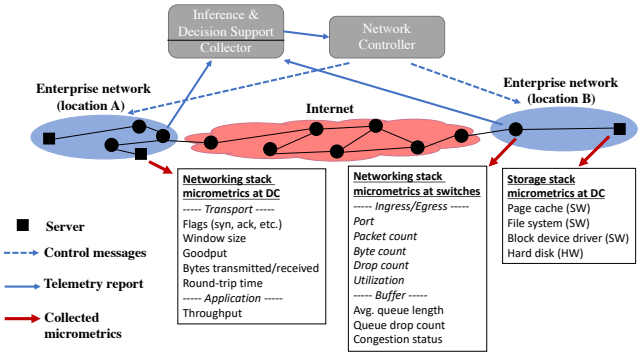

Figure 1: **High-level MicroMon Design.** Figure shows the integration of our HW and SW micro-metrics collection and dissemination in MicroMon.

4.1.4 MicroMon Components

We develop MicroMon, a replacement of Cassandra's Snitch mechanism for fine-grained micrometrics collection and multi-dimensional resource monitoring as shown in Figure 1. As discussed earlier in § 2, in Cassandra, each node gets assigned to some key range and act as a coordinator as well as replica for other set of keys. Hence, in each node, a user-level component—*MicroMon-engine*—integrated with Cassandra, is responsible for collecting micrometrics from its own node as well as replicas. In addition to the user-level component, MicroMon also contains an OS-level driver (*MicroMon-driver*) to monitor hardware micrometrics (e.g., SMART counters [23] such as high P/E warnings, NIC packet drops) and software micrometrics (e.g., application's page cache state, storage and network I/O queue delays, TCP queue occupancy). Further, our choice of extending snitch as opposed to designing a new tool is mainly because of Snitch's wide usage and to avoid polluting replicas with new monitoring tools.

4.2 Micrometrics Collection & Dissemination

We next focus on designing techniques to reduce micrometrics collection and dissemination overheads, since collection and dissemination happens at both hosts and the network.

4.2.1 Host-level collection with Anomaly Reports

One issue complicates the collection of micrometrics at the host level. Specifically, the number of micrometrics increases with the number of hardware resources and software subsystems used by an application. For example, even a single SSD's SMART counters contain close to 32 counters that can impact performance [23]. This, coupled with software micrometrics, could significantly impact the resolution at which this information is collected and disseminated, micrometrics in data plane at high resolution could impact impacting the performance of applications. To address the issue, we propose anomaly reports, where for all possible host-level micrometrics, the host OS only reports anomalies. Such reports require no further processing at the inference and decision engine (explained below). We argue that host OSes can already identify software and hardware anomalies, given a better holistic view of the system. For example, monitoring anomalies such as high page cache misses, network queue latency or high SSD P/E count can be done at the host, only reporting anomalies rather than processing them. Further, we propose to extend the *MicroMon-driver* to report such anomalies. *MicroMon-driver* periodically collects the required software and hardware micrometrics and composes a monitoring packet with anomalies.

4.2.2 Co-designing Network-level Dissemination with Host OSes

We leverage the innovations in programmable switches and co-design two mechanisms (with host OSes) to extend network telemetry to support heterogeneous resource monitoring. Our first mechanism (a) generates monitoring packets using MicroMon-driver with anomalies identified at hosts as payload and (b) uses in-band network telemetry (INT), subsequently, to add network micrometrics (identified in § 4.1.3) to those monitoring packets and disseminate them via sink nodes to the decision engine. In a general INT model, data packets contain headers which in turn contain instructions for the traffic sources (e.g., applications, end-host networking stacks) about what state to collect and write into the packets as it transits the network. However, given the limitations of INT packet metadata sizes and INT instruction fields (16-bits), and the possibility of hundreds of micrometrics, we leverage anomaly reports and using techniques such as run-length encoding and memoization for the ones collected at the network. Telemetry reports can be generated at switches based on pre-established anomalies, opening up the possibility of reporting only aggregated network events and naturally overcoming the micrometrics collection and dissemination. For geo-distributed deployments as shown in Figure 1, we plan to leverage INT over any encapsulation (e.g., over TCP/UDP, depending on the application) to collect and disseminate micrometrics.

Our second mechanism is very similar to the first one, except the usage of INT in the dissemination step above. Specifically, programmable switches also support out-of-band reporting of telemetry reports directly to the decision engine. This is to overcome the difficulties in strategic placement of INT sink nodes in an enterprise WAN. In light of this, we support out-of-band telemetry reports containing aggregated network and host micrometrics to be reported directly to decision engine.

4.3 Scalable Inference Logic

Next, we discuss our preliminary scoring-based inference in Cassandra and then discuss our ongoing work on building a generalized inference logic.

Scoring-based Inference. The current snitching mechanism in Cassandra uses a scoring mechanism that sorts the endpoints by their latency with an adaptive replica failure detec-

tor. The coordinator node initially sorts the replicas based on their network proximity and then uses response latency to update the score frequently. Instead, in MicroMon, we modify the scoring mechanism to consider different software and hardware storage and network micro-metrics in addition to straight forward network latency and bandwidth. Our current prototypic scoring mechanism (as evaluated), assigns equal weights to all software and hardware micro-metrics and higher weights to network latency and bandwidth.

Ongoing: Generic Inference System in MicroMon. We are currently re-designing the simple scoring-based inference system to create a framework in a generic, data-driven, multi-objective optimization that not only collects micro-metrics but also to produce a ranks stable and/or performant nodes. When the marginal utility of the micro-metrics collected is low (or below the Pareto front of the analysis objectives), those micro-metrics can be excluded. For example, for a delay-sensitive operation, the collection of compute metrics can be relaxed with minimal or no impact on the application's performance.

5 Experimental Evaluation of MicroMon

To understand the performance benefits and implication of our proposed MicroMon, we compare the performance of MicroMon with the vanilla dynamic snitching mechanism in Cassandra, under different deployments, by varying multidimensional micrometrics such as storage heterogeneity (SSD vs. HDD), application threads, network latencies across replicas, and the page cache.

Benchmark. We run the industry standard and widely-used YCSB [8] benchmark with Cassandra. Due to space restrictions, we study three key-value access patterns from the YCSB cloud suite: (1) Workload A (a write-heavy workload with 50/50 read-write ratio), Workload B with 95% read, and Workload C (a read-only workload).

Experimental Setup. We design our experiments in the CloudLab infrastructure with three physical nodes located within the Utah and Utah-APT clusters. For the SSD replica located in the Utah cluster, we use a system with a 2.0GHz, 8-core Intel Xeon D-1548 processor and 64 GB ECC memory. For the other two replicas residing in the Utah-APT cluster, their systems entail a 2.1GHz, 8-core Intel Xeon E5-2450 processor with 16 GB RDIMM memory. The keys are mapped across these three Cassandra nodes, where each node acts as a coordinator for a range of keys with other nodes acting as a replica. Two out of three Cassandra nodes use a hard-disk drive with 1-2 ms random-access latency and 5-15 MB/s random access bandwidth compared to the SSD node with 10-30 us latency and 300 MB/s bandwidth. The average network latency between nodes solely within Utah-APT is around 0.25-0.35 ms latency, and between nodes from Utah-APT and Utah is around 0.35-0.45ms latency. In all our experiments, we compare MicroMon against the default dynamic snitching mechanism that mainly considers network latency.

5.1 Impact of Heterogeneous Storage

First, to understand the impact of considering underlying storage hardware's latency and bandwidth, we compare the default dynamic-snitching mechanism in Cassandra against MicroMon. For avoiding undue overheads of network latency, we maintain two replicas of Cassandra at the same datacenter, with one replica using SSD and other replica using hard-disk to serve YCSB requests. We maintain Cassandra's in-memory buffer (skiplist) size to the default 32MB and commit the log (using *fsync*) every 10ms (a parameter in Cassandra).

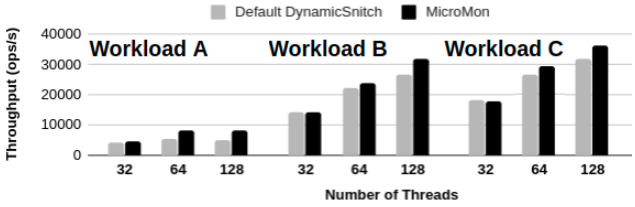

Figure 2: **Storage hardware impact.** Results for YCSB workloads varying the number of threads and the storage hardware across replicas.

Figure 2 shows the throughput of YCSB workloads A, B, and C in the y-axis, and the number of client threads issuing requests on the x-axis. The client issue 1KB (YCSB's default request size) requests totaling 500K operations. First, the dynamic snitch in Cassandra is oblivious to storage latencies and only considers network latencies. Due to the lack of network latencies, it just sends most requests to both SSD and HDD replicas. In contrast, MicroMon also considers storage latency through the micro-metric collection, redirecting a significant number of requests to SSD replica. Workload A with 50% read and writes shows higher gains over workload B with 5% writes. Further, with increasing client threads (in the x-axis), the MicroMon gains improve by exploiting SSDs higher parallelism, leading to 40% gains for workload A. Finally, for read-only workload C combined with the YCSB's uniform distribution, the benefits are lower for our current evaluation scale. Also, note that we only report the run-phase and not the warmup phase (cold access). *The results highlight the need for multi-dimension micrometrics, which includes storage hardware monitoring.*

5.2 Network Latency and Page Cache

To understand the combined impact of storage and network heterogeneity, in Figure 3, we compare the Default DynamicSnitch with our MicroMon based on synthesized latency toward the SSD replica, whereas the Utah-APT replicas' network latency (<1ms) is unchanged. In the x-axis, we gradually increase the network latency of SSD from 0*ms* and 25*ms*. We include two types of comparisons with respect to the SSD node, one where the cache remains as default and the other where the SSD replica has a lower page cache footprint (MicroMon No Cache and Default DynamicSnitch No Cache) simulated by frequently clearing the cache. For brevity, we

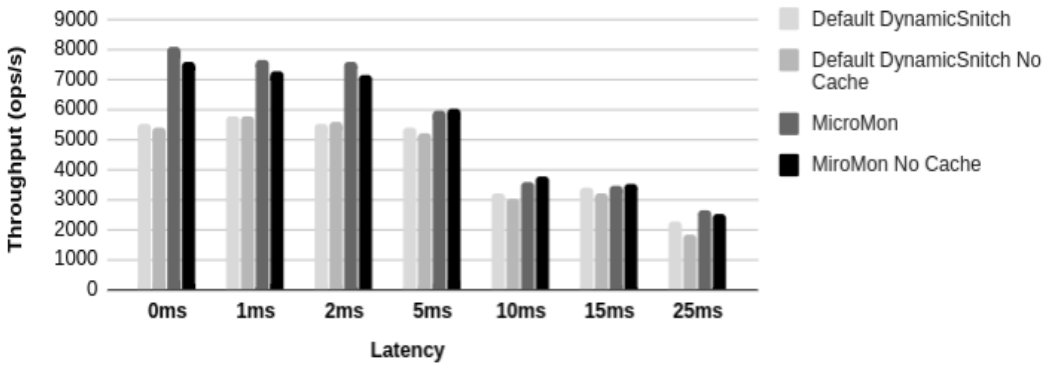

Figure 3: **Network and Storage Latency Impact.** Results show the combined considering network and storage latency for YCSB workload A with 16-threads.

only show YCSB's workload A (50% write and read) for 64 threads.

First, without any added network latency, the SSD replica in a remote cluster provides a significant throughput (40% increase; see "Default DynamicSnitch" and "MicroMon" bars in Figure 3) compared to the Utah-APT replicas. However, when adding 5ms latency to the SSD replica's network latency, the throughput reduces when compared to the Utah-APT replicas, but still maintaining a considerable (10% increase) throughput increase. Next, regarding the page cache, for the SSD replica with lower page-cache (see "No Cache" bars), we notice that even without network latency, MicroMon prioritizes the Utah-APT nodes a little more than the default DynamicSnitch resulting in slightly higher average throughput (37% increase; see "Default DynamicSnitch No Cache" and "MicroMon No Cache" bars in Figure 3). In case of increase in network latencies, we observe a marginal decrease in throughput gains. For example, while we observe a throughput gain of 33% using MicroMon for 2ms network latency, the throughput gain decreases to 24% in case of 25ms of latency. These results highlight the need for multidimensional monitoring.

6 Summary and Future Work

We present MicroMon, a multi-dimensional monitoring and inference framework for geo-distributed applications using heterogeneous hardware. We introduce micrometrics, which is a set of fine-grained hardware and software metrics required to study the combined impact of heterogeneous WAN and storage resources on Cassandra's performance. To reduce micrometrics collection and dissemination overheads, we propose anomaly reports and concerted effort between the programmable switches and host OSes to reduce the overhead of collecting and disseminating thousands of micrometrics in WAN. Our prototype deployed in a geo-distributed Cassandra using heterogeneous storage and network shows close 49% performance gains.

Future Work. Only capturing currently available hardware and software micrometrics (for network and storage) is insufficient. This calls for consideration of broader class of (multi-dimensional) resources (e.g., compute, memory, network, storage). Further, identifying and innovating new microarchitecture-level micrometrics that capture the impact of multi-dimensional resource usage holistically is critical. For example, no micrometrics exists to capture the hardware overheads in using the same PCI channels for a storage-intensive and network-intensive application. We believe a cross-stack fine-grained heterogeneous monitoring of memory, storage, network, and compute requires microarchitecture-level innovations in the hardware and new software micrometrics. We plan to consider microarchitectural innovations for MicroMon as part of future work. In addition, the decision of what micrometrics to use must be automated without relying on administrators, application developers or network operators.

Finally, as noted in § 4.3, the current prototype of MicroMon assigns equal weights to all software and hardware micrometrics. Unfortunately, this is not applicable for all the applications. For example, micrometrics for bandwidth-sensitive applications require higher weights for micrometrics collected from the WAN than those collected from the end hosts. Profiling the weights micrometrics for diverse applications atop heterogeneous resources is a complex problem. We intend to consider this in MicroMon as part of future work.

Acknowledgements

We thank the anonymous reviewers and our Shepherd, Malte Schwarzkopf, for their insightful feedback. This work is supported by NSF CNS 1910593 and NSF CNS 1850297 awards. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF.

References

[1] Amazon CloudWatch. https://aws.amazon.com/cloudwatch/.

[2] Apache Cassandra. http://cassandra.apache.org/.

[3] Azure Monitor. https://docs.microsoft.com/en-us/azure/azure-monitor/overview/.

[4] In-band Network Telemetry (INT). https://p4.org/assets/INT-current-spec.pdf.

[5] Linux SNMP Counter. https://www.kernel.org/doc/html/latest/networking/snmp_counter.html.

[6] Snitch. http://cassandra.apache.org/doc/4.0/operating/snitch.html/.

[7] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[8] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, Indianapolis, Indiana, USA, 2010.

[9] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3), August 2013.

[10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, volume 41, pages 205–220. ACM, 2007.

[11] Lautaro Dolberg, Jerome Francois, and Thomas Engel. Efficient multi-dimensional aggregation for large scale monitoring. In *Presented as part of the 26th Large Installation System Administration Conference (LISA 12)*, pages 163–180, San Diego, CA, 2012. USENIX.

[12] Monia Ghobadi and Ratul Mahajan. Optical layer failures in a large backbone. In *Proceedings of the 2016 Internet Measurement Conference*, pages 461–467. ACM, 2016.

[13] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4):455–466, August 2014.

[14] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[15] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Rethinking adaptability in wide-area stream processing systems. In *Proceedings of the 10th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'18, pages 2–2, Berkeley, CA, USA, 2018. USENIX Association.

[16] Sergey Legtchenko, Hugh Williams, Kaveh Razavi, Austin Donnelly, Richard Black, Andrew Douglas, Nathanael Cheriere, Daniel Fryer, Kai Mast, Angela Demke Brown, Ana Klimovic, Andy Slowey, and Antony Rowstron. Understanding rack-scale disaggregated storage. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, Santa Clara, CA, July 2017. USENIX Association.

[17] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 311–324, Santa Clara, CA, March 2016. USENIX Association.

[18] Jianzhe Liang, Jun Bi, Yu Zhou, and Cheng Zhang. In-band network function telemetry. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, page 42–44, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Jason Mars and Lingjia Tang. Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 619–630, New York, NY, USA, 2013. ACM.

[20] Jason Mars and Lingjia Tang. Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers. *SIGARCH Comput. Archit. News*, 41(3):619–630, June 2013.

[21] Justin Moore and Jeff Chase. Data center workload monitoring, analysis, and emulation. In *in Eighth Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2005.

[22] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. Ssd failures in datacenters: What? when? and why? In *Proceedings of the 9th ACM International on Systems and Storage Conference*, SYSTOR '16, pages 7:1–7:11, New York, NY, USA, 2016. ACM.

[23] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. Ssd failures in datacenters: What? when? and why? In *Proceedings of the 9th ACM International on Systems and Storage Conference*, SYSTOR '16, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 361–378, Boston, MA, February 2019. USENIX Association.

[25] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *5th USENIX Conference on File and Storage Technologies (FAST 07)*, San Jose, CA, February 2007. USENIX Association.

[26] Diana Andreea Popescu, Noa Zilberman, and Andrew William Moore. Characterizing the impact of network latency on cloud-based applications' performance. 2017.

[27] Zhufan Wang, Guangyan Zhang, Yang Wang, Qinglin Yang, and Jiaji Zhu. Dayu: Fast and low-interference data recovery in very-large storage systems. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 993–1008, Renton, WA, July 2019. USENIX Association.

[28] Hobin Yoon, Juncheng Yang, Sveinn Fannar Kristjansson, Steinn E. Sigurdarson, Ymir Vigfusson, and Ada Gavrilovska. Mutant: Balancing storage cost and latency in lsm-tree data stores. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, pages 162–173, New York, NY, USA, 2018. ACM.

[29] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 29–42, Lombard, IL, 2013. USENIX.