Research Statement

Mingzhe Hao, University of Chicago

As numerous new data-intensive applications and storage hardware emerge, maintaining performance sustainability and robustness of data and storage systems is becoming more intricate and challenging. Users want numerous demands (e.g., real-time latency, continuously high throughput, workload elasticity) to be met. Service providers are facing a hard task of delivering acceptable service-level objectives (SLOs) such as low and highly stable latencies. Both parties essentially wish for the same goal, but the gap in between continous to widen tragically and become more complex. Customers keep introducing more data paradigms (e.g., big data, machine learning, IoT) and bombing providers with application-specific requirements that are more than non-trivial to fulfill, which brings a growing threat to designing generic systems that can persistently deliver rapid performance.

My research aims at building fast and robust next-generation data and storage systems. Specifically, I architect these systems generically to achieve rapid responses of low latency even in the most turmoil scenarios. As systems grow in complexity, my approach is to tackle this significant problem from different angles:

1. **Data approach [6]**: We should have a thorough and scaled understanding of real-world issues with increasing complicacy to help us pinpoint the potential crux and solutions. One of my works mines performance logs tracking half a million disks and thousands of SSDs, and to the best of my knowledge is the most extensive study of storage device-level performance variability.

2. **Hardware-level approach [11, 13]**: While other approaches attempt to reduce performance variability at the application level with approaches like speculation, I see a different point of view, whereas cutting performance variability "at the source" is more effective. Specifically, we re-architect SSDs that can exhibit "tiny" tail latencies and also flash arrays that collaborate with the host and circumvent almost all noises induced by background operations.

3. **OS-level approach [5, 9]**: At the heart of the system stack is the OS; hence, the question is how the OS should evolve today to provide stable performance for the deep stack. In tackling this problem, my insight is that the OS is not just the OS for personal computers, but rather the OS for the "datacenter". In this context, I have designed an OS that is SLO-aware and capable of predicting every IO latency and failing over slow IOs to peer OSs.

4. **ML-for-system approach [7, 12]**: Current systems are growing too complex for human designers to come up with a heuristic-based policy for optimal system control. So many different storage models exist, which are very heterogeneous with performance unpredictability. Applications cannot reason about how they work, and predicting systems' performance is a black art. This situation raises the question of whether machine learning can help. While others have used machine learning to predict storage performance (but in a coarse-grained way), here, I take a much deeper dive to use neural networks to predict the performance of every request and every IO, making unforeseeable systems performance highly predictable.

My multi-angle approach has allowed me to have a vast horizon of the challenge, which makes my solutions well accepted by the community. My works and collaborations on under-studied performance faults from large-scale systems/services have motivated numerous research projects from vendors, operators, and system designers. The hardware-level studies I am involved in are nominated as the best papers for their potential impacts. The OS-level designs I advocate can outperform the industrial mechanisms deployed at Google. The ML-for-system approach I have been working on has been widely tested on multiple production settings and has shown considerable performance advantages. Finally, I am happy to say that my works have been accepted and presented in top system/storage conferences such as ASPLOS, FAST, OSDI, SoCC, and SOSP.

1 Major Research Projects

Data approach *I advocate that the surging amount of logs collected at the production frontline is a perfect indicator for future challenges, cruxes, and potential solutions.*

As storage has been growing enormously over the past decade with the large-scale adoption of SSDs, modern applications particularly demand low and predictable response times, giving rise to stringent performance SLOs at the millisecond level. To get a quantified understanding of the performance variance in their systems and to mitigate potential SLO violations, many companies have been collecting performance logs at fine granularity for individual storage devices, thus, creating highly valuable knowledge bases that are far from fully explored. Given this opportunity, I worked with NetApp and conducted a systematic, large-scale study of performance in over 450,000 disks and 4,000 SSDs for millions of drive hours in total [6]. Specifically, for every single drive in every hour, I checked how slow the drive performed compared with its peers and correlated these slowdowns with other metrics, including workload density, drive event, age, and model. The result clearly reflects the pervasiveness of performance instability. We found that severe slowdown (at least 2× slower than peers) is widespread and destructive: (1) 26% of disks and 29% of SSDs have experienced this 2× slowdown; (2) Disks and SSDs are slow for 0.22% and 0.58% of the drive hours; (3) These slowdown occurrences can silently continue for hours; (4) A slow drive can significantly hinder an entire aggregation unit (e.g. RAID). Further investigation revealed that, unfortunately, no external signal could strongly indicate these slowdowns, as they are mostly caused by internal idiosyncrasies of hardware devices. This discovery, together with the statistical trend shown by the data that SSDs (and other advanced hardware, as illustrated in [4]) are more likely to be unstable, highly incentivizes the development of hardware-level solutions.

Hardware-level approach *I advocate that exposures of internal information and control features (white-box/gray-box) can enable powerful proactive solutions that bring significant performance advantages.*

Motivated by the previous study on performance variance, I participated in the ttFlash project [13], which aims at eliminating the performance tail caused by the notorious garbage collection (GC) process. As a hardware-level work, ttFlash proposes a GC-tolerant SSD that can deliver and guarantee stable performance by circumventing GC-blocked IOs. Enlightened by the evolvement of SSD internal technology, we leverage three major technological advancements for building ttFlash: (1) Increasing power and speed of today's flash controllers that support more complex logic; (2) Redundant Array of NAND (RAIN), a standard data protection mechanism that allows for circumventing blocked reads with parity regeneration; (3) A large RAM buffer to mask write tail latencies from GC operations. The timely combination of above technology practices gives new advantages to ttFLash, which dramatically reduces the number of blocked IOs from 2-7% (base case) to only 0.003-0.7% and GC-induced slowdown from 5.6-138.2× to 1.0-2.6×.

With a similar principle, I helped with building a tail-evading flash array delivering predictable performance by using a fast-fail approach to circumvent busy sub-IOs [11]. While ttFlash requires a thorough grasp of internal knowledge (white-box), which is not a desirable option for many commodity SSDs as it requires "open" devices, this work only needs limited control (gray-box) without adding a new interface or modifying the heavy SSD internal management. With the support from the extended NVMe command, OS/host layer, and IO Determinism (IOD) interface, the flash array controller can proactively and deterministically reconstruct late requests with data redundancy (e.g., parity in RAID). As a result, ttFlash comes significantly close to a "no-GC" scenario. Specifically, between 99-99.99th percentiles, ttFlash is only 1.0-2.6× slower than the no-GC case, while a base approach suffers from 5-138× GC-induced slowdowns. Compared to other works, our mechanism is more deterministic in trimming the tail latencies and easy to deploy with minimal changes.

OS-level approach *I advocate that transparency through layers/components can considerably improve the efficiency and flexibility of the modern composite system stack.*

One of my works, MittOS [5], offers operating system support on performance transparency so applications can customize their failover mechanisms according to their multifarious SLOs and application settings. MittOS exposes

a fast-rejecting SLO-aware interface for applications to exploit so users can provide their SLOs and conduct fast failover when the SLOs cannot be fulfilled. Different from existing proactive tail-tolerant approaches such as cloning, tied requests, and snitching, MittOS does not need to introduce extra IO density or complex revocation management in the application layer and can work effectively even when the noise is bursty in millisecond scale. Compared with "reactive" mechanisms that pay an expensive wait in the "millisecond era," MittOS's "no-wait" approach can dramatically outperform. An evaluation on distributed NoSQL systems shows that MittOS can reduce IO completion time up to 35% compared to the latest wait-then-speculate approaches.

Another project (named Leap [9]) I joined promotes address transparency across components in the cloud storage stack to smooth the offload of complex storage services to today's IO accelerators. Nowadays, the cloud storage stack is extremely hungry. Cloud providers must pay 10-20% of x86 cores to maintain these functions. Ideally, these functions are better offloaded to IO accelerators so that host CPU cores can be spent on customer VMs. However, current IO accelerators are not designed for end-to-end scenarios or offloading complex storage functions. To resolve this real-deployment challenge, Leap employs a set of OS/software techniques on top of hardware capabilities to provide a uniform address space across x86 cores and IO accelerators, allowing the host to portably leverage the accelerators. Evaluations on a Leap prototype with low-cost ARC SoCs and x86 hosts indicate that Leap is "offload ready," as this setting can bring more than 20$\times$ cost saving with only 2-5% overhead on local/remote storage services.

ML-for-system approach *I advocate that increasingly powerful heuristics are yet not enough to catch up with the exploding growth of the design space, which calls for help from machine learning techniques.*

Even though modern data systems are becoming too complex to get a whole picture, heuristics can still help extract useful information and build robust solutions. One of my recent works [12] probes the internal properties of commercial SSDs, which are among the most complicated hardware components. Although these SSDs are usually treated as black-boxes, heuristic-based methods can extract some critical internal properties, turning these devices from "black" to "gray." Based on a "common-knowledge" understanding of the SSD internal structure, this work develops an application-level tool that can cover multiple properties for any block-device SSDs. The result of running this tool on various SSD models reflects numerous observations and unique findings, exposing substantial improvement space for both SSD users and manufacturers.

Nevertheless, unfortunately, heuristics are still far from getting omniscient for SSDs, not to mention other complex components. Internal idiosyncrasies such as dynamic FTL policies and GC mechanisms remain as key obstacles to precise performance prediction at finer-granularity and optimal proactive tail-tolerance solutions for flash-based storage. To tackle this challenge of refining prediction from the previous aggregate level (e.g., average latency/throughput) to the individual level (e.g. the latency of each IO), I introduce a simple, yet effective machine learning mechanism to complement my heuristic-based solutions in [5] and [12]. Based on the available input in block-device layer and drivers, with appropriate feature settings, a lightweight neural network can help to accurately predict the occurrences of slow IOs with $<$ 10us preprocess and inference overhead on CPU. By associatively combining the indications of ML and heuristic (e.g., #pending IOs, the timings of buffer flushes, etc.), this hybrid solution can promptly forecast the late IOs with 99%+ accuracy on production read-only traces and 89% on read-write ones.

2 Other Research Projects

Besides my major works on improving storage performance, I researched other system aspects such as reliability, availability and QoS. I have applied my data approach to (1) analyzing public bug repositories with 3000+ "vital" issues in cloud systems [2] and (2) studying news and reports that detail unplanned outages from popular Internet services [3]. At hardware-level, I have helped develop a flash emulator to facilitate SSD research [10]. Additionally, I have applied my OS-/system-level approach to (1) understand the impact of "limping" hardware in cloud systems [1] and (2) use system-domain semantics to accelerate distributed system model checkers [8].

3 Future Research

In the future, I would like to continue my effort in developing ML-based solutions for data and storage systems ("***ML-for-storage***"). Also, I am interested in building storage for various machine learning scenarios ("***Storage-for-ML***"). For both directions, I plan to stick with my current data, hardware-level, and OS-/system-level approaches.

ML-for-storage

Data approach ML-for-storage is an under-explored and under-guided field. Though there is an increasing amount of related works in the recent couple of years, these works, created by pioneer researchers, are still relatively unorganized, as many open questions emerge together with the encouraging results. How should we apply the large collection of ML mechanisms? On which components? For what policies? At which granularity (e.g. aggregate/real-time)? How do we prove their effectiveness and applicability? How about existing heuristic-based approaches? Should ML completely replace them? Or should they co-exist? To which extent? I believe a continuous, iterative, and large-scale study of existing works can help us clarify these confusions and figure out the best way to integrate ML into our storage stacks.

Hardware-level approach Nowadays, hardware storage devices can be generally split into two classes: black-box ones (e.g., regular commercial SSDs) and white/gray-box ones (e.g., software-defined flash and multi-stream SSDs, which give users more control). Both types have their disadvantages: black-box devices are hard to support optimal performance given little knowledge about upper layers, and white/gray-box ones require applications to clearly and correctly define the data access patterns. With increasing computing power inside the hardware (e.g., powerful inherent processor insides the SSDs), our devices can adopt ML (perhaps lightweight) to help dynamically adjust its internal policy to adapt to workloads of different types. I believe this is an inspiring direction to look into, especially since the internal processors can have more access to the hardware details, which brings strategic advantages.

OS-/System-level approach Motivated by the preliminary success of [7], which manages to promptly run ML on CPU to guide ms-/us-level real-time operations. I believe that there is a lot more that ML can do for OS-/system-level solutions. We can use ML to help revise application behaviors; we can run ML to assist OS in policy-making; we can choose to utilize ML at coarse and aggregate levels (e.g., every several seconds/minutes), and we can exploit it for real-time decisions. I am very excited to explore this broad open research area of ML-for-systems.

Storage-for-ML

Data approach Given the exploding growth of AI, machine learning would probably be among the significant running tasks with numerous settings and diverse purposes. Each combination will leave a unique data footprint pattern on storage, which is critical for the performance of these ML runtimes. A firm grasp of how data is accessed on broad-scale production ML setups can provide tremendous insights on how to customize our storage to help ML run rapidly and smoothly.

Hardware-level approach Hinted by the benefits of allowing IO accelerators to directly talk to storage devices, as mentioned in [9], I believe the regular chain of "storage devices - memory - CPU - GPU" may not fit the interests of all ML scenarios. Some users may prefer GPU to directly access fast flash-based storage to avoid contentions on CPU and memory resources; some users may want to utilize the computing power of storage devices for ML, while others may want to see how ML runs with small-capacity CPU and NVRAM in small portable devices. The correlation between storage hardware and computing units is still uncertain, and it is interesting to know.

OS-/System-level approach Current software components for storage management (e.g., file systems) are mostly of general purposes. Apparently, they may not work well for all types of work (for example, database systems prefer to manage their own buffer), including machine learning. Similar to database systems, which access a massive amount of data, ML applications may also want a particular route to data self-management for their various data operations such as batching/shuffling. I feel that ML, as a gigantic data consumer, deserves to get some VIP treatments from the storage side.

References

[1] Thanh Do, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, and Haryadi S. Gunawi. Limplock: Understanding the Impact of Limpware on Scale-Out Cloud Systems. In *Proceedings of the 4th ACM Symposium on Cloud Computing (SoCC)*, 2013.

[2] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC)*, 2014.

[3] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC)*, 2016.

[4] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *Proceedings of the 16th USENIX Symposium on File and Storage Technologies (FAST)*, 2018.

[5] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[6] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *Proceedings of the 14th USENIX Symposium on File and Storage Technologies (FAST)*, 2016.

[7] Mingzhe Hao, Levent Toksoz, Edward Edberg, Andrew A. Chien, Henry Hoffmann, and Haryadi S. Gunawi. [Machine Learning for Operating Systems: A Case of Using Deep Neural Network for OS-Level I/O Latency Prediction]. In *Preparation*, 2020.

[8] Tanakorn Leesatapornwongsa, Mingzhe Hao, Pallavi Joshi, Jeffrey F. Lukman, and Haryadi S. Gunawi. SAMC: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems. In *Proceedings of the 11th Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[9] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. Efficient and Portable Virtual NVMe Storage on ARM SoCs. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[10] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In *Proceedings of the 16th USENIX Symposium on File and Storage Technologies (FAST)*, 2018.

[11] Huaicheng Li, Ronald Shi, Mingzhe Hao, Martin L. Putra, Fadhil I. Kurnia, Achmad I. Kistijantoro, Sujin Park, Jaeyoung Do, Xing Lin, and Haryadi S. Gunawi. [Title Intentionally Blinded Due to Potential Conflicts]. In *Submission to the Proceedings of the 18th USENIX Symposium on File and Storage Technologies (FAST)*, 2020.

[12] Nanqinqin Li, Mingzhe Hao, Xing Lin, Huaicheng Li, Levent Toksoz, Tim Emami, and Haryadi S. Gunawi. [Title Intentionally Blinded Due to Potential Conflicts]. In *Submission to the Proceedings of the 18th USENIX Symposium on File and Storage Technologies (FAST)*, 2020.

[13] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *Proceedings of the 15th USENIX Symposium on File and Storage Technologies (FAST)*, 2017.