Fail-Slow at Scale

Evidence of Hardware Performance Faults in Large Production Systems

HARYADI S. GUNAWI, RIZA O. SUMINTO, RUSSELL SEARS, SWAMINATHAN SUNDARARAMAN, XING LIN, AND ROBERT RICCI

Haryadi Gunawi is a Neubauer Family Assistant Professor in the Department of Computer Science at the University of Chicago where he leads the UCARE Lab (U Chicago systems research on Availability, Reliability, and Efficiency). He received his PhD from the University of Wisconsin—Madison and was awarded an Honorable Mention for the 2009 ACM Doctoral Dissertation Award. haryadi@cs.uchicago.edu

Riza Suminto received a BS in computer science from Gadjah Mada University in 2010. In 2013, he joined the University of Chicago to pursue his PhD in computer science. He is currently a member of the UCARE Lab and is interested in addressing performance and outage bugs in cloud systems. riza@cs.uchicago.edu

Russell Sears is a Senior Engineer at Pure Storage. His research interests include high-performance and scalable systems with a focus on storage infrastructure. He is currently working on new storage APIs to replace flash translation layers and the block device abstraction. sears@purestorage.com

Swaminathan (Swami) Sundararaman is the Lead Architect of ParallelM, an early-stage startup focused on production machine learning and deep learning. Swami was previously at Fusion-io Inc. and Sandisk Corp. He holds a PhD from the University of Wisconsin—Madison. swaminathan.sundararaman@parallelmachines.com

Understanding fault models is an important criterion for building robust systems. Decades of research have developed mature failure models such as fail-stop [10], fail-partial [2], fail-transient [9], and Byzantine failures [5]. We highlight an under-studied "new" failure type: fail-slow hardware, i.e., hardware that is still running and functional but in a degraded mode, i.e., slower than its expected performance. We found that all major hardware components can exhibit fail-slow faults. For example, disk throughput can drop by three orders of magnitude to 100 KB/s due to vibration; CPUs can unexpectedly run at half-speed due to lack of power; and network card performance can collapse to Kbps level due to buffer corruption and retransmission.

While fail-slow hardware arguably did not surface frequently in the past, in today's systems, deployed at scale along with many intricacies of large-scale operational conditions, the probability that a fail-slow hardware incident can occur increases. Furthermore, as hardware technology continues to scale (smaller and more complex), today's hardware development and manufacturing will only exacerbate the problem.

To fill the void of strong evidence of hardware performance faults in the field, we—a group of researchers, engineers, and operators of large-scale datacenter systems across 12 institutions—decided to write this "community paper" [11]. More specifically, we have collected 101 detailed reports of fail-slow hardware behaviors, including the hardware types, root causes, symptoms, and impacts to high-level software.

Methodology

We collected 101 reports of fail-slow hardware from large-scale cluster deployments in 12 institutions (Table 1). At such scales, hardware is more likely to witness fail-slow occurrences. The reports were all unformatted text, written by the engineers and operators who still vividly remember the incidents due to the severity of the impacts. The incidents were reported between 2000 and 2017, with only 30 reports predating 2010. Each institution reported a unique set of root causes. For example, although an institution may have seen a corrupt buffer as the root cause slowing down networking hardware (packet loss and retransmission) many times, it was only collected as one report. Thus, a single report can represent multiple instances of an incident. If multiple institutions report the same root cause,

Institution	Nodes
Company 1	>10,000
Company 2	150
Company 3	100
Company 4	>1,000
Company 5	>10,000
University A	300
University B	>100
University C	>1,000
University D	500
Nat'l Lab X	>1,000
Nat'l Lab Y	>10,000
Nat'l Lab Z	>10,000

Table 1: Operational scale

Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems

Xing Lin is a member of the technical staff in the NetApp Advanced Technology Group. He joined NetApp after receiving his PhD from the University of Utah, where his research focused on improving space efficiency for storage systems. xing.lin@netapp.com

Robert Ricci is a Research Associate Professor in the School of Computing at the University of Utah and is one of the directors of the Flux Research Group. He has been a designer and implementer of research infrastructure for nearly two decades, including at Emulab and CloudLab. ricci@cs.utah.edu

however, it is counted multiple times. The majority of root causes (66%) were unique, however, and only 22% were duplicates (12% of the reports did not pinpoint a root cause). More specifically, a duplicated incident was reported on average by 2.4 institutions; for example, firmware bugs were reported from five institutions, driver bugs from three institutions, and the remaining issues from two institutions. The raw (partial) data set can be downloaded on our group website [1].

We note that there are no analyzable hardware-level performance logs (more in the To Vendors section, below), which prevents large-scale log studies. We strongly believe that there were many more cases that slipped by unnoticed. Some occurrences undoubtedly went unreported since operators change jobs. We did not include known slowdowns (e.g., random I/Os causing slow disks, or GC activities occasionally slowing down SSDs). We only include reports of unexpected degradation. For example, unexpected hardware faults that make GC activities work harder are reported.

Important Findings and Observations
Varying root causes: Fail-slow hardware can be induced by internal causes such as firmware bugs or device errors/wear-outs as well as external factors such as configuration, environment, temperature, and power issues.
Faults convert from one form to another: Fail-stop, -partial, and -transient faults can convert to fail-slow faults (e.g., the overhead of frequent error masking of corrupt data can lead to performance degradation).
Varying symptoms: Fail-slow behavior can exhibit a permanent slowdown, transient slowdown (up-and-down performance), partial slowdown (degradation of sub-components), and transient stop (e.g., occasional reboots).
A long chain of root causes: Fail-slow hardware can be induced by a long chain of causes (e.g., a fan stopped working, making other fans run at maximal speeds, causing heavy vibration that degraded the disk performance).
Cascading impacts: A fail-slow hardware can collapse the entire cluster performance; for example, a degraded NIC made many jobs lock task slots/containers in healthy machines, hence new jobs cannot find enough free slots.
Rare but deadly (long time to detect): It can take hours to months to pinpoint and isolate a fail-slow hardware for many reasons (e.g., no full-stack visibility, environment conditions, cascading root causes and impacts).
Suggestions
To vendors: When error masking becomes more frequent (e.g., due to increasing internal faults), more explicit signals should be thrown rather than running with a high overhead. Device-level performance statistics should be collected and reported (e.g., via SMART) to facilitate further studies.
To operators: Thirty-nine percent of root causes are external; thus troubleshooting fail-slow hardware must be done online. Due to cascading root causes and impacts, full-stack monitoring is needed. Fail-slow root causes and impacts exhibit some correlation; thus statistical correlation techniques may be useful (with full-stack monitoring).
To systems designers: While software systems are effective in handling the fail-stop (binary) model, more research is needed to tolerate fail-slow (non-binary) behavior. System architects, designers, and developers can fault-inject their systems with all the root causes reported in this study to evaluate the robustness of their systems.

Table 2: Summary of our findings and suggestions

Hardware Types						
Root	**SSD**	**Disk**	**Mem**	**Net**	**CPU**	**Total**
Device errors	10	8	9	10	3	40
Firmware bugs	6	3	0	9	2	20
Temperature	1	3	0	2	5	11
Power	1	0	1	0	6	8
Environment	3	5	2	4	4	18
Configuration	1	1	0	2	3	7
Unknown	0	3	1	2	2	8
Total	22	23	13	29	25	112

Table 3: Root causes across hardware types. "Unknown" implies that operators could not pinpoint the root cause but simply replaced the hardware. Note that a report can have multiple root causes (e.g., environment and power/temperature issues), and thus the total (112) is larger than the 101 reports.

Observations (Take-Away Points)

Varying Root Causes

Pinpointing the root cause of a fail-slow hardware is a daunting task as it can be induced in a variety of ways, as shown in Table 3. Hardware performance fault can be caused by *internal* root causes from within the device such as *firmware issues* or *device errors/wear-outs*. For example, many individual I/Os in SSD that should only take tens of µs were throttled by exactly multiples of 250µs, as high as 2-3ms; a bad batch of SSDs stopped responding for seconds and then recovered; a disk head moved slower due to gunk that spilled from the actuator assembly and accumulated between the disk head and the platter; and a NIC driver bug caused a "very poor" throughput, and the operators had to disable TCP offload to work around the problem.

However, a perfectly working device can also be degraded by many *external* root causes such as *configuration*, *environment*, *temperature*, and *power*-related issues. Some examples of external causes are: a clogged air filter caused optics in the switch to start failing due to a high temperature, generating a high 10% packet-loss rate; a partial power supply failure meant throttling the CPUs by 50%; some nodes were running slow because other nodes in the same rack were drawing more power, causing rack power supply instability and dropping power to various parts of the rack; and faulty chassis fans surrounding nodes caused such a strong vibration that drives went into recovery mode.

Fault Conversions to Fail-Slow

Different types of faults such as fail-stop, -partial, and -transient can convert to fail-slow faults.

Fail-stop to fail-slow: Because many hardware pieces are connected, a fail-stop component can make other components exhibit a fail-slow behavior. For example, a dead power supply throttled the CPUs by 50% since the backup supply was unable to deliver enough power; and a vendor's buggy firmware made a batch of SSDs stop for seconds, disabling the flash cache layer and slowing the entire storage stack. These examples suggest that fail-slow occurrences can be correlated to other fail-stop faults in the system. A robust fail-stop-tolerant system should ensure that a fail-stop fault does not convert to fail-slow.

Fail-transient to fail-slow: In addition to fail-stop, many kinds of hardware can exhibit fail-transient errors: for example, disks occasionally return I/O errors, processors sometimes produce a wrong result, and memory corrupts from time to time. Due to their transient and "rare" nature, firmware/software typically masks these errors from users. A simple mechanism is to *retry* the operation or *repair* the error (e.g., with ECC or parity). When the transient failures are recurring much more frequently, however, *error masking* can be a "double-edged sword." That is, because error masking is not a free operation (there are retry delays, repair costs), when the errors are not rare, the masking overhead becomes the common case performance.

We observed many cases of fail-transient to fail-slow conversion. For example, a disk firmware triggered frequent "read-after-write" checks in a degraded disk; and many cases of loss/corrupt network packets (a 1–50% rate in our reports) triggered heavy retries that collapsed the network throughput by orders of magnitude.

From the stories above, it is clear that a distinction must be made between rare and frequent fail-transient faults. While it is acceptable to mask the former, the latter should be exposed to and not hidden from high-level software stack and monitoring tools.

Fail-partial to fail-slow: Some hardware can also exhibit fail-partial fault where only some part of the device is unusable (that is, a partial fail-stop). This kind of failure is typically masked by the firmware/software layer (e.g., with remapping). However, when the scale of partial failure grows, the fault masking brings a negative impact to performance. Bad chips in SSDs reduce the size of over-provisioned space, triggering more frequent garbage collection; and a more known problem, remapping a large number of bad sectors, can induce more disk seeks. Similar to the fail-transient case above, there must be a distinction between small- and large-scale partial faults.

Varying Fail-Slow Symptoms

We observed the "many faces" of fail-slow symptoms: permanent, transient, and partial fail-slow and transient fail-stop, as illustrated in Figure 1.

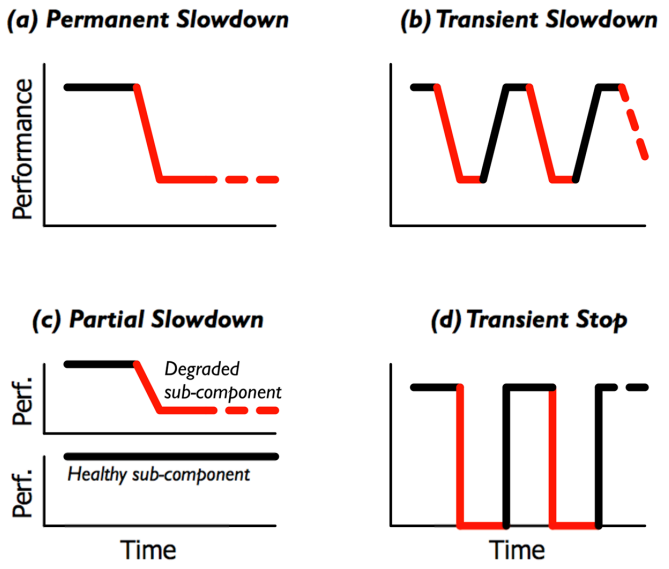

Figure 1: Fail-slow symptoms. The figure shows four types.

Permanent slowdown: The first symptom (Figure 1a) is a permanent slowdown, wherein the device initially works normally but its performance drops off over time and does not return to the normal condition (until the problem is manually fixed). This mode is the simplest among the four models because operators can consistently see the issue.

Transient slowdown: The second symptom (Figure 1b) is a transient slowdown, wherein the device's performance fluctuates between normal and significant degradation, which is more difficult to troubleshoot. For example, applications that create a massive load can cause the rack power control to deliver insufficient power to other machines (degrading their performance), but only until the power-hungry applications finish.

Partial slowdown: The third model (Figure 1c) is a partial slowdown, where only some parts of the device slow. In other words, this is the case of partial fail-stop converting to partial slowdown. For example, some parts of memory that are faulty require more ECC checks to be performed. The partial fail-slow model also complicates debugging since some operations experience the slowdown but others on the same device are not affected.

Transient stop: The last symptom (Figure 1d) is the case of transient stop, where the device occasionally reboots itself, causing performance at times to degrade to zero. For example, a buggy firmware sometimes made the SSDs "disappear" from the RAID controller and later reappear.

Cascading Causes and Impacts

Another intricacy of fail-slow hardware is the chain of cascading events: First, between the actual root cause and the hardware's fail-slow symptom, there is a chain of *cascading root causes*.

Second, the fail-slow symptom then creates *cascading impacts* to the high-level software stack, and potentially to the entire cluster.

Here are some examples of long cascading root causes that lead to fail-slow hardware. A fan in a compute node stopped working, making other fans compensate for the dead fan by operating at maximal speeds, which then caused a lot of noise and vibration that subsequently degraded the disk performance. When a piece of hardware becomes fail-slow, not only does it affect the host machine, but it can cause cascading impacts across the cluster. For example, a degraded NIC in one machine, slowing from 1 Gbps to 1 Kbps, caused a chain reaction that slowed down the entire cluster of 100 machines as the connecting tasks that were affected held up containers/slots for a long time, and new jobs could not run due to the slot shortage.

Rare but Deadly: Long Time-to-Detect

The fail-slow hardware incidents in our report took *hours* or even *months* to detect (pinpoint). More specifically, 1% of the cases were detected in minutes, 13% in hours, 13% in days, 11% in weeks, 17% in months, with an unknown time in 45% of cases. Some engineers called this a "costly debugging tail." In one incident, an entire team of engineers was pulled to debug the problem, costing the institution tens of thousands of dollars. There are several reasons why the time-to-detect (TTD) is long.

First, the fact that the incidence of fail-slow hardware is not as frequent as fail-stop cases implies that today's software systems do not completely anticipate (that is, undermine) such scenarios. Thus, while more-frequent failures can be solved quickly, less-frequent but more complex failures (that cannot be mitigated by the system) can significantly cost the engineers time.

Second, as explained before, the root cause might not originate from the fail-slow hardware. For example, the case of transient slowdown caused by power-hungry applications took months to figure out since the problem was not rooted in the slow machines nor the power supply.

Third, external-environment conditions beyond the control of the operators can prolong diagnosis. For months, a vendor failed to reproduce the fail-slow symptoms in its sea-level testing facility since the hardware only slowed down at a high mountain altitude.

Finally, operators do not always have full visibility of the entire hardware stack. For example, an incident took days to solve because the operators had no visibility into the power supply health.

Suggestions

In addition to cataloging instances of fail-slow hardware, a goal of this study is to offer vendors, operators, and systems designers insights about how to address this poorly studied failure mode.

To Vendors

Making implicit error masking explicit: Fail-slow hardware can be categorized as an "implicit" fault, meaning it does not always return any explicit hard errors (e.g., due to error masking; see the Fault Conversions to Fail-Slow section, above). However, there were many cases of slowly increasing error rates that would eventually cause cascading performance failures. Vendors might consider throwing explicit error signals when the error rates far exceed the expected rate.

Exposing device-level performance statistics: Modern hardware now exposes such information via SMART. However, our conversations with operators suggest that the information from SMART is "insufficient to act on." We hope vendors will expose device-level performance data to support future statistical studies.

To Operators

Online diagnosis: In our study, 39% of the cases were caused by external root causes. Some reports suggest that operators took days or even months to diagnose the problem since it could not be reproduced in offline testing. Thus, online diagnosis is important, but also not straightforward, because not all hardware components are typically monitored, which we will discuss next.

Monitoring of all hardware components: Today, in addition to main hardware components, other hardware components and environment conditions such as fan speeds and temperature are also monitored. Unfortunately, not all hardware is monitored in practice. For example, multiple organizations failed to monitor network cables, using the flow of traffic as a proxy for cable health instead. The diagnosis took much longer because blame for poor performance is usually directed towards the main hardware components such as NICs or switches. The challenge is then to prevent too much data being logged.

Another operational challenge is that different teams are responsible for different parts of the data center: software behavior, machine performance, cooling, power. With limited views, operators cannot fully diagnose the problem.

A future challenge relates to a proprietary full-packaged solution like hyper-converged or rack-scale design. Such design usually comes with the vendor's monitoring tools, which might not monitor and expose all information to the operators. Instead, vendors of such systems often monitor hardware health remotely, which can lead to fragmentation of monitoring infrastructure as the number of vendors increases.

Correlating full-stack information: With full-stack performance data, operators can use statistical approaches to pinpoint and isolate the root cause [6].

Although most of the cases in our study were hard-to-diagnose problems, the revealed root causes were relatively "simple." For example, when a power-hungry application was running, it drained the rack power and degraded other nodes. Such a correlation can easily be made but requires processing of power-level information. Future research can be done to evaluate whether existing statistical monitoring approaches can detect such correlations.

While the metrics above are easy to monitor, there are other fine-grained metrics that are hard to correlate. For example, in one configuration issue, only multicast network traffic was affected, and in another similar one, only big packets (>1500 bytes) experienced long latencies. In these examples, the contrast between multicast and unicast traffics and small and big packets is clear. However, to make the correlation, detailed packet characteristics must be logged as well.

Finally, monitoring algorithms should also detect "counter-intuitive" correlations. For example, when user performance degrades, operators tend to react by adding more nodes. However, there were cases where adding more nodes did not translate to better performance since the underlying root cause was not isolated.

To Systems Designers

While the previous section focuses on post-mortem remedies, this section provides some suggestions on how to better anticipate fail-slow hardware in future systems.

Making implicit error-masking explicit: Similar to hardware, error masking (as well as "tail" masking) in higher software stacks can make the problem worse. We have observed fail-slow hardware that caused many jobs to time out and be retried again repeatedly, consuming many other resources and converting the single hardware problem into larger cluster-wide failures. Software systems should not just silently work around fail-slow hardware but need to expose enough information to help troubleshooting.

Fail-slow to fail-stop: Earlier, we discussed many fault conversions to fail-slow faults. The reverse can be asked: can fail-slow faults be converted into fail-stop mode? Such a concept is appealing because modern systems are well equipped to handle fail-stop failures [3]. We next discuss opportunities and challenges of this concept.

Skip non-primary fail-slow components: Some resources, such as caching layers, can be considered nonprimary components. For example, in many deployments, SSDs are treated as a caching layer for the back-end disks. The assumption that SSD is always fast and never stalls does not always hold. Thus, when fail-slow SSDs (acting as a caching layer) introduce more latencies than the back-end disks, they can be skipped temporarily until the problem subsides. However, consistency issues must be taken into account. Another suggestion is to run in "partial"

mode rather than in full mode but with slow performance. For example, if many disks cause heavy vibration that degrades the disk throughput significantly, it is better to run fewer disks to eliminate the throughput-degrading vibration [4].

Detect fail-slow recurrences: Another method to make slow-to-stop conversion is to monitor the recurrence of fail-slow faults. For example, when disks or SSDs continue to "flip-flop" between online/offline mode, triggering RAID rebalancing all the time, it is better to take them offline. We observed several cases of transient fail-slow hardware that was taken offline, but after passing the in-office diagnosis, the device was put online again, only to cause the same problem.

Fail-slow fault injections: System architects can inject fail-slow root causes reported in this study to their systems and analyze the impacts.

One can argue that asynchronous distributed systems (eventual consistency) should naturally tolerate fail-slow behaviors. While this is true, there are many stateful systems that cannot work in fully asynchronous mode: in widely used open-sourced distributed systems, fail-slow hardware can cause cascading failures such as thread pool exhaustion, message backlogs, and out-of-memory errors [8].

Tail-tolerant distributed systems [7] are supposed to be resilient. However, other recent work shows that the "tail" concept only targets performance degradation from resource contention, which is different from the fail-slow hardware model such as slow NICs; as a result, not all tail-tolerant systems, like Hadoop or Spark, can cut tail latencies induced by degraded NICs [13].

Beyond networking components, the assumption that storage latency is stable is also fatal. It has been reported that disk delays cause race conditions or deadlock in distributed consistency protocols [12]. The problem is that some consistency protocols, while tolerating network delays, do not incorporate the possibility of disk delays, for the sake of simplicity.

With fail-slow injections, operators can also evaluate whether their systems or monitoring tools signal the right warnings or errors. There were a few cases in our reports where wrong signals were sent, causing the operators to debug only the healthy part of the system.

Overall, we strongly believe that injecting root causes reported in this study will reveal many flaws in existing systems. Furthermore, all forms of fail-slow hardware such as slow NICs, switches, disks, SSD, NVDIMM, and CPUs need to be exercised since they lead to different symptoms. The challenge is then to build future systems that enable various fail-slow behaviors to be injected easily.

Conclusion

Today's software systems are arguably robust at logging and recovering from fail-stop hardware—there is a clear, binary signal that is fairly easy to recognize and interpret. We believe fail-slow hardware is a fundamentally harder problem to solve. It is very hard to distinguish such cases from ones that are caused by software performance issues. It is also evident that many modern, advanced deployed systems do not anticipate this failure mode. We hope that our study can influence vendors, operators, and systems designers to treat fail-slow hardware as a separate class of failures and start addressing them more robustly in future systems.

Complete List of Authors

Haryadi S. Gunawi, Riza O. Suminto, Mingzhe Hao, and Huaicheng Li, University of Chicago

Russell Sears and Casey Golliher, Pure Storage

Swaminathan Sundararaman, Parallel Machines

Xing Lin and Tim Emami, NetApp

Weiguang Sheng and Nematollah Bidokhti, Huawei

Caitie McCaffrey, Twitter

Gary Grider and Parks M. Fields, Los Alamos National Laboratory

Kevin Harms and Robert B. Ross, Argonne National Laboratory

Andree Jacobson, New Mexico Consortium

Robert Ricci and Kirk Webb, University of Utah

Peter Alvaro, University of California, Santa Cruz

H. Birali Runesha, Mingzhe Hao, and Huaicheng Li, University of Chicago Research Computing Center

References

[1] Download the fail-slow database: http://ucare.cs.uchicago.edu/projects/failslow/.

[2] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," in *Proceedings of the 2007 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007: http://research.cs.wisc.edu/adsl/Publications/latent-sigmetrics07.pdf.

[3] G. Candea and A. Fox, "Crash-Only Software," in *Proceedings of the Ninth Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.

[4] C. S. Chan, B. Pan, K. Gross, K. Gross, T. S. Rosing, and K. Vaidyanathan, "Correcting Vibration-Induced Performance Degradation in Enterprise Servers," in *Proceedings of the Greenmetrics workshop (Greenmetrics)*, 2013: http://seelab.ucsd.edu/papers/cschan_gm13.pdf.

[5] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults," in *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI)*, 2009: https://www.usenix.org/legacy/event/nsdi09/tech/full_papers/clement/clement.pdf.

[6] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang, "PerfScope: Practical Online Server Performance Bug Inference in Production Cloud Computing Infrastructures," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC)*, 2014: http://www.nipunarora.net/pdf/perfscope.pdf.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, 2004: https://www.usenix.org/legacy/event/osdi04/tech/full_papers/dean/dean.pdf.

[8] T. Do, M. Hao, T. Leesatapornwongsa, T. 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: http://ucare.cs.uchicago.edu/pdf/socc13-limplock.pdf.

[9] T. Do, T. Harter, Y. Liu, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "HardFS: Hardening HDFS with Selective and Lightweight Versioning," in *Proceedings of the 11th USENIX Symposium on File and Storage Technologies (FAST)*, 2013: https://www.usenix.org/system/files/conference/fast13/fast13-final70_0.pdf.

[10] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. 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: http://ucare.cs.uchicago.edu/pdf/socc16-cos.pdf.

[11] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. 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 '16)*, 2018: https://www.usenix.org/system/files/conference/fast18/fast18-gunawi.pdf.

[12] T. Leesatapornwongsa, J. F. Lukman, S. Lu, and H. S. Gunawi, "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016: http://ucare.cs.uchicago.edu/pdf/asplos16-TaxDC.pdf.

[13] R. O. Suminto, C. A. Stuardo, A. Clark, H. Ke, T. Leesatapornwongsa, B. Fu, D. H. Kurniawan, V. Martin, U. M. Rao G., and H. S. Gunawi, "PBSE: A Robust Path-Based Speculative Execution for Degraded-Network Tail Tolerance in Data-Parallel Frameworks," in *Proceedings of the 8th ACM Symposium on Cloud Computing (SoCC)*, 2017: http://ucare.cs.uchicago.edu/pdf/socc17-pbse.pdf.