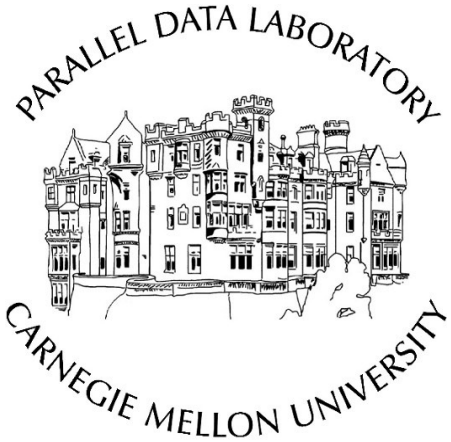

Automation without predictability is a recipe for failure

Raja R. Sambasivan & Gregory R. Ganger

CMU-PDL-11-101
January 2011

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Automated management seems a must, as distributed systems and datacenters continue to grow in scale and complexity. But, automation of performance problem diagnosis and tuning relies upon predictability, which in turn relies upon low variance—most automation tools aren't effective when variance is regularly high. This paper argues that, for automation to become a reality, system builders must treat variance as an important metric and make conscious decisions about where to reduce it. To help with this task, we describe a framework for understanding sources of variance and describe an example tool for helping identify them.

Acknowledgements: We thank the members and companies of the PDL Consortium (including APC, EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi, IBM, Intel, LSI, Microsoft Research, NEC Laboratories, NetApp, Oracle, Riverbed, Samsung, Seagate, STEC, Symantec, VMWare, and Yahoo! Labs) for their interest, insights, feedback, and support. This research was sponsored in part by a Google research award, NSF grants #CNS-0326453 and #CCF-0621508, by DoE award DE-FC02-06ER25767, and by CyLab under ARO grants DAAD19-02-1-0389 and W911NF-09-1-0273.

Keywords: automated management, automation, autonomic computing, datacenters, distributed systems, performance problem diagnosis, predictability, variance

1 Introduction

Many in the distributed systems community have recognized the need for automated management in datacenters and large distributed systems [9, 15]. They predict that the rapidly increasing scale and complexity of these systems will soon exceed the limits of human capability. Automation is the only recourse, lest they become completely unmanageable. In response to this call to arms, there have been many research papers published on tools for automating various tasks, especially performance diagnosis [3, 6, 7, 13, 14, 17, 20, 21, 24]. Most focus on layering automation on top of existing systems, simply assuming that these systems exhibit a key property needed for automation—predictability, or low variance. Many systems do not, especially the most complex ones that need automation the most. As such, this assumption severely limits both the utility of existing automation tools and the scope of tasks that can be automated.

Our own experiences using an automated performance diagnosis tool, Spectroscope [20], to diagnose problems in distributed storage systems, such as Ursa Minor [1] and Bigtable [5], bear out the inadequacy of the low-variance assumption. Though useful, Spectroscope has been unable to reach its true potential due to high variance resulting from poorly structured code, high resource contention, and hardware issues.

The quality of automation could be improved by encouraging system builders to engineer systems that exhibit low variance. In areas where predictability is the most important metric, this unilateral policy might be the right approach. For example, in the early 1990s, the US Postal Service decided consistency of mail delivery times was more important than raw speed and slowed down mail delivery. When asked why this tradeoff was made, the postmaster general responded: "I began to hear complaints from mailers and customers about inconsistent first-class mail delivery...We learned how important consistent, reliable delivery is to our customers." [4]. In scientific computing, inter-node variance can drastically limit performance due to frequent synchronization barriers. In real-time systems, it is more important for programs to meet each deadline than run faster on average.

Of course, in many cases, variance is a side effect of desirable performance enchancements. Caches, a mainstay of most distributed systems, intentionally trade variance for performance. Many scheduling algorithms do the same [25, 26]. Additionally, reducing variance blindly may lead to synchronized "bad states," which may result in failures or drastic performance problems. For example, Phanishayee et al. describe a phenomenon, known as TCP Incast, in which synchronized reads in a cluster-storage system can overload the intermediary switch, resulting in throughput as low as 1–10% of the client's bandwidth capacity [19].

Some variance in distributed systems is unavoidable. For example, Arpaci-Dusseau et al. show that identical components, such as disks from the same vendor, can differ significantly in performance due to fault-masking techniques [2]. Also, it may be difficult to design complex systems to exhibit low variance because it is hard to predict their precise operating conditions [11, 16].

In practice, there is no easy answer in deciding how to address variance to aid automation. There is, however, a wrong answer—ignoring it, as is being done today. Instead, for the highly-touted goal of automation to become a reality, system builders must treat variance as a *first-class metric*. They should use detailed instrumentation to localize the highest sources of variance in their systems and make conscious decisions about whether to reduce or eliminate their variance. They should account for desirable high variance sources by explicitly marking them to help automation tools isolate their variance from other areas of the system, thus increasing the portions for which good predictions can be made.

The rest of this paper is organized as follows. Section 2 discusses how automated performance diagnosis tools suffer from high variance. Section 3 identifies three types of variance that can be found in distributed systems and how each should be addressed. Section 4 proposes a mechanism system builders can use to identify the sources of variance in their system. Section 5 describes open questions, and Section 6 concludes.

2 The impact of high variance

Tools that automate aspects of performance diagnosis all require low variance in various metrics to perform well. When important metrics exhibit high variance, false positives and/or false negatives will increase, as the tool struggles to distinguish normal performance variations from problematic ones. False positives, or mispredictions, perhaps represent the worst failure mode due to human effort wasted.

Most performance diagnosis tools use a combination of three techniques to make predictions; their failure modes depend on which ones they use. Tools that make predictions when certain metrics exceed thresholds are the most unpredictable. They will yield more false positives or false negatives when variance is high, depending on the exact value of the threshold. A low threshold will result in more false positives, whereas increasing it to accommodate the high variance will mask problems, resulting in more false negatives.

To avoid costly mispredictions, a few statistical tools account for how variance affects the expected false-positive rate of their predictions. Some choose not to predict when the expected rate exceeds a pre-set one (e.g., 5%). The cost of high variance for them is an increase in false negatives. Others use adaptive techniques to try to increase confidence before making predictions, either by increasing the sampling rate or by obtaining more data. The cost of high variance for these tools is increased storage/processing cost and an increase in time required before predictions can be made. The specific adaptive technique used—ad-hoc or algorithmic—differs depending on the tool, with an unfortunate bias toward the former.

Regardless of the failure mode, the quality of the results returned by any automated performance diagnosis tool will increase as variance is reduced. The rest of this section describes how four tools from recent literature are affected by variance. Table 1 provides a summary and lists additional tools.

Magpie [3]: This tool uses an unsupervised clustering algorithm to identify anomalous requests in a distributed system. Requests are grouped together based on similarity in request structure and resource usage; small clusters are identified as anomalies. A threshold is used to decide whether to place a request in the cluster deemed most similar to it, or whether to create a new one. High variance will yield many small clusters if the threshold value is low, resulting in an increase in false positives. Combating high variance by increasing the threshold will result in more false negatives.

Spectroscope [20]: This tool uses statistical techniques to automatically identify the *mutations* in timing responsible for an observed performance change between two periods in a distributed storage system. It relies on the expectation that requests that take the same path through a distributed storage system should incur similar performance costs. High variance in this metric will increase the number of false negatives. This tool also uses a threshold to identify mutations in request structure that contributed to the performance change, such as requests that used to hit in cache, but which now miss. High variance in request structure between periods will result in more false positives or false negatives, depending on the threshold chosen.

Peer comparison [13, 14]: These diagnosis tools are intended to be used on tightly coupled distributed systems, such as Hadoop and PVFS. They rely on the expectation that every machine in a given cluster should exhibit similar behaviour. As such, they indict a machine as exhibiting a performance problem if its performance metrics differ significantly from others. Thresholds are used to determine the degree of difference tolerated. High variance in metric distributions between machines will result in more false positives, or false negatives, depending on the threshold chosen.

Tool described by Oliner [17]: This tool identifies correlations in anomalies across components of a distributed system. To do so, it first calculates an *anomaly score* for discrete time intervals by comparing the distribution of some signal—e.g., average latency—during the interval to the overall distribution. The strength of this calculation is dependent on low variance in the signal. High variance will yield lower scores, resulting in more false negatives. The author himself states this fact: "The [anomaly] signal should usually take values close to the mean of its distribution—this is an obvious consequence of its intended semantics..." [17].

Tool	FPs / FNs	Tool	FPs / FNs
Magpie	⇑ / ⇑	DARC [17]	- / ⇑
Spectroscope	⇑ / ⇑	Pinpoint [6]	- / ⇑
Peer comp.	⇑ / ⇑	Shen [21]	- / ⇑
Oliner	- / ⇑	SLIC [7]	- / ⇑

Table 1: **How the predictions made by automated performance diagnosis tools are affected by high variance.** High variance will yield more false positives (FPs) or false negatives (FNs), depending on the techniques used by the tool to make predictions. Note that the tools shown on the right are not described in the text.

3 Three types of variance

Variance in distributed systems is an important metric that directly affects potential for automation. To reduce it, two complementary courses of action are necessary. During the design phase, system builders need to make conscious decisions about which areas of the distributed system should be more predictable (display low variance). Since the complexity of distributed systems makes it unlikely they will be able to identify all of the sources of variance during design [2, 11, 16], they must also work to identify sources of variance during development and testing. To help with the latter, this section describes a nomenclature for variance sources that can help system builders reason about them and understand which ones' variance should be reduced.

Intentional variance sources: These are a result of a conscious tradeoff made by system builders. For example, such variance may emanate from a scheduling algorithm that lowers mean response time at the expense of variance. Alternatively, it may result from explicit anti-correlation added to a component to prevent it from entering synchronized, stuck states (e.g., Microreboots [18]). Labeling a source as intentional indicates the system builder will not try to reduce its variance.

Inadvertent variance sources: These are often the result of poorly designed or implemented code; as such, their variance should be reduced or eliminated. For example, such variance sources may include functions that exhibit extraordinarily varied response times because they contain many different control paths (spaghetti code). In [13], Kasick el al. describe how such high variance functions were problematic for an automated diagnosis tool developed for PVFS. Such variance can also emanate from unforeseen interactions between components, or may be the result of contention for a resource. The latter is interesting in that it suggests that certain performance problems can be diagnosed directly by localizing variance. In fact, while developing Spectroscope [20], we found that high variance was the best predictor of problems caused by contention.

Unavoidable variance sources: These are contained in areas of the distributed system that cannot be easily modified or replaced. Examples may include non-flat network topologies within a datacenter or disks that exhibit high variance in bandwidth between their inner and outer zones [2]. Labeling a source as unavoidable indicates that the system builder does not have the means to reduce its variance.

Variance from intentional and unavoidable sources may be given, so the quality of predictions made by automation tools in these areas will suffer. However, it is important to guarantee their variance does not impact predictions made for other areas of the system. This may be the case if the data granularity used by an automated tool to make predictions is not high enough to distinguish between a high variance source and surrounding areas. For example, problems in the software stack of a component may go unnoticed if an automated tool does not distinguish it from a high-variance disk. To avoid such scenarios, system builders should help automated tools account for high variance sources directly—for example, by adding markers around them that are used by automated tools to increase their data granularity.

4 VarianceFinder

To illustrate a variance-oriented mindset, this section proposes one potential mechanism, called VarianceFinder, for helping system builders identify the main sources of variance in their systems during development and testing. The relatively simple design outlined here focuses on reducing variance in response times for distributed storage systems such as Ursa Minor [1], Bigtable [5], and GFS [10]. However, we believe this basic approach could be extended to include other performance metrics and other types of systems.

VarianceFinder utilizes end-to-end traces (Section 4.1) and follows a two-tiered approach. First, it shows the variance associated with aspects of the system's overall functionality that should exhibit similar performance (Section 4.2). Second, it allows system builders to select functionality with high variance and identifies the components, functions, or RPCs responsible, allowing them to take appropriate action (Section 4.3). We believe this tiered approach will allow system builders to expend effort where it is most needed.

4.1 End-to-end tracing

To identify sources of variance within a distributed system, a fine-grained instrumentation mechanism is needed. End-to-end tracing satisfies this requirement, as it captures the detailed control flow of individual requests within and across the components of a distributed system. Many implementations exist, all of which are relatively similar [3, 8, 23]. Figure 1 shows an example *request-flow graph* generated from Stardust [23], the end-to-end tracing mechanism used in Ursa Minor [1]. Note that nodes in this graph indicate trace points reached by the request, whereas edges are annotated with performance metrics—in this case the latency between executing successive trace points. Trace points are automatically inserted within the system's middleware (e.g., RPC layer), but should also be manually added at key points of interest within the system.

End-to-end tracing incurs very little overhead (around 1% with request-level sampling [20, 22]) and is gaining traction in production systems, such as Google datacenters [22]. We believe it forms an excellent basis on top of which tools for identifying sources of variance can be built.

4.2 Id'ing functionality & first-tier output

To identify functionality that should exhibit similar performance, VarianceFinder utilizes an informal expectation, common in distributed storage systems, that requests that take the same path through the system should incur similar performance costs. For example, system builders generally expect READ requests whose metadata and data hit in an NFS server's cache to perform similarly. VarianceFinder groups request-flow graphs that exhibit the same structure—i.e., those that represent identical activities and execute the same trace points—into *categories* and calculates average response times, variances, and squared coefficients of variation (C^2) for each. C^2, which is defined as $(\frac{\sigma}{\mu})^2$, is a normalized measure of variance and captures the intuition that categories whose standard deviation is much greater than the mean are worse offenders than those whose standard deviation is less than or close to the mean. In practice, categories with $C^2 > 1$ are said to have high variance around the mean, whereas those with $C^2 < 1$ exhibit low variance around the mean [12]. Figure 2 illustrates this process.

The first-tier output from VarianceFinder consists of the list of categories ranked by C^2 value. System builders can click through highly-ranked categories to see a graph view of the request structure, allowing them to determine whether it is important. For example, a highly-ranked category that contains READ requests likely will be deemed important, whereas one that contains rare requests for the names of mounted volumes likely will not.

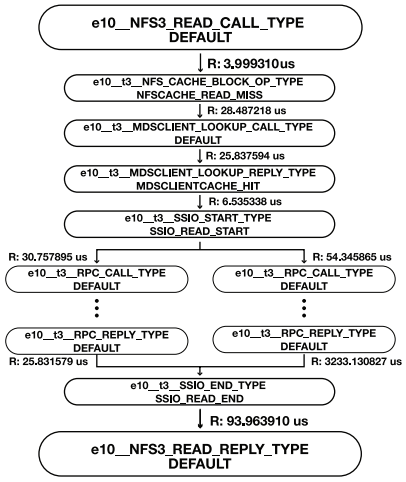

Figure 1: **Example request-flow graph.** The graph shows a striped READ in the Ursa Minor [1] distributed storage system. Nodes represent trace points and edges are labeled with the time between successive events. Node labels are constructed by concatenating the machine name (e.g., `e10`), component name (e.g., `NFS3`), trace point name (e.g., `READ_CALL_TYPE`), and an optional semantic label (e.g., `NFSCACHE_READ_MISS`). Due to space constraints, trace points executed on other components as a result of the NFS server's RPC calls are not shown.

4.3 Second-tier output & resulting actions

Once the system builder has selected an important highly-ranked category, he can use VarianceFinder to localize its main sources of variance. This is done by highlighting the highest-variance edges along the critical path of the category's requests. In some cases, an edge may exhibit high variance because of another edge—for example, an edge spanning a queue might display high variance because the component to which it sends data also does so. To help system builders understand these dependencies, clicking on a highlighted edge will reveal other edges that have non-zero covariance with it.

Knowing the edges responsible for the high variance allows the system builder to investigate the relevant areas of the system. Variance from sources that he deems inadvertent should be reduced or eliminated. Alternatively, he might decide that a high-variance source is intentional, or unavoidable. In such cases, he should add additional, tight trace points around the source to serve as markers. Automation tools that use these markers to increase their data granularity—especially those that use end-to-end traces directly [3, 20, 22]—will be able to make better predictions about areas surrounding the high-variance source.

Adding instrumentation can also help reveal previously unknown interesting behaviour. The system builder might decide that an edge exhibits high variance because it encompasses too large of an area of the system, merging many dissimilar behaviours. In such cases, extra trace points should be added to disambiguate them.

5 So, how much can be done?

This paper argues that variance needs to be addressed explicitly during design and implementation of distributed systems, if automated management is to become a reality. But, much research is needed to understand how much variance can and needs to be reduced, the difficulty of doing so, and the resulting reduction in management effort.

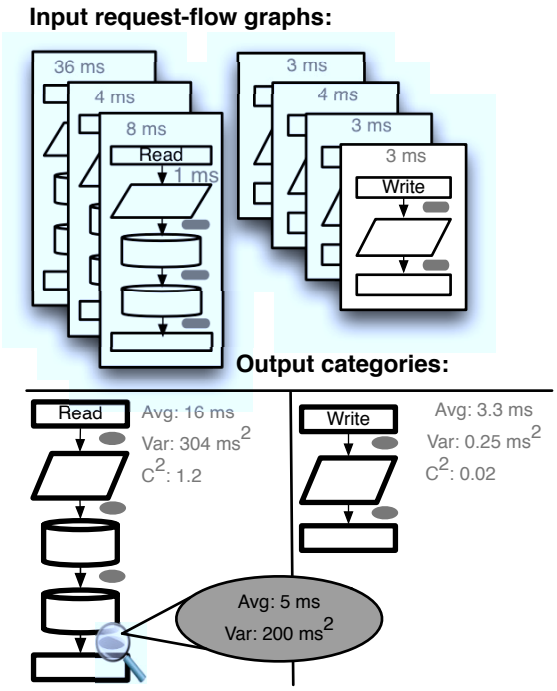

Figure 2: **Example of how a VarianceFinder implementation might categorize requests to identify functionality with high variance.** VarianceFinder assumes that requests that take the same path through a distributed system should incur similar costs. It groups request-flow graphs that exhibit the same structure into *categories* and calculates statistical metrics for them, including the squared coefficient of variation (C^2).

To answer the above questions, it is important that we work to identify the breakdown of inadvertent, intentional, and unavoidable variance sources in distributed systems and datacenters. If unavoidable sources are the largest contributors, hardware manufacturers, such as disk drive vendors, may also have to incorporate variance as a first-class metric and strive to minimize it. The benefits of better automation must be judged by how real people utilize and react to automation tools, not via simulated experiments or fault injection.

There is an often-said proverb: "nothing worth doing is easy." This is true of automation as well. Answering the questions posed above will help us understand if the effort necessary to automate management is indeed worth it, or whether continuing along and focusing on the path of manual management is the correct course.

6 Conclusion

Though automation in large distributed systems is a desirable goal, it cannot be achieved when variance is high. This paper presents a framework for understanding types of variance sources and a potential mechanism for identifying them. We imagine that there are many other tools and design patterns for reducing variance and enhancing predictability. In the interim, those building automation tools must consider whether the underlying system exhibits the variance properties needed for success.

References

[1] Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, and Jay J. Wylie. Ursa Minor: versatile cluster-based storage. *Conference on File and Storage Technologies* (San Francisco, CA, 13–16 December 2005), pages 59–72. USENIX Association, 2005.

[2] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. Fail-stutter fault tolerance. *Hot Topics in Operating Systems* (Schloss Elmau, Germany, 21–23 May 2001), pages 33–37. IEEE, 2001.

[3] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using Magpie for request extraction and workload modelling. *Symposium on Operating Systems Design and Implementation* (San Francisco, CA, December 2004), pages 259–272. USENIX Association, 2004.

[4] James Bovard. Slower is better: the new postal service. *Individual Liberty, Free Markets, and Peace*, **146**, 1991. http://www.cato.org/pubs/pas/pa-146.html.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. *Symposium on Operating Systems Design and Implementation* (Seattle, WA, 06–08 November 2006), pages 205–218. USENIX Association, 2006.

[6] Mike Y. Chen, Anthony Accardi, Emre Kiciman, Dave Patterson, Armando Fox, and Eric Brewer. Path-based failure and evolution management. *Symposium on Networked Systems Design and Implementation* (San Francisco, CA, 29–31 March 2004), pages 309–322. USENIX Association, 2004.

[7] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. *ACM Symposium on Operating System Principles* (Brighton, United Kingdom, 23–26 October 2005), pages 105–118. ACM, 2005.

[8] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-Trace: a pervasive network tracing framework. *Symposium on Networked Systems Design and Implementation* (Cambridge, MA, 11–13 April 2007). USENIX Association, 2007.

[9] Gregory R. Ganger, John D. Strunk, and Andrew J. Klosterman. *Self-* Storage: brick-based storage with automated administration*. Technical Report CMU–CS–03–178. Carnegie Mellon University, August 2003.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. *ACM Symposium on Operating System Principles* (Lake George, NY, 10–22 October 2003), pages 29–43. ACM, 2003.

[11] Steven D. Gribble. Robustness in Complex Systems. *Hot Topics in Operating Systems* (Schloss Elmau, Germany, 21–23 May 2001), pages 21–26. IEEE, 2001.

[12] Mor Harchol-Balter. 15-857, Fall 2009: Performance modeling class lecture notes, 2009. http://www.cs.cmu.edu/~harchol/Perfclass/class09fall.html.

[13] Michael P. Kasick, Rajeev Gandhi, and Priya Narasimhan. Behavior-based problem localization for parallel file systems. *Hot Topics in System Dependability* (Vancouver, BC, Canada, 03–03 October 2010). USENIX Association, 2010.

[14] Michael P. Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Black-box problem diagnosis in parallel file systems. *Conference on File and Storage Technologies* (San Jose, CA, 24–26 February 2010). USENIX Association, 2010.

[15] Jeffrye O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, **36**(1):41–50. IEEE, January 2003.

[16] Jeffery C. Mogul. Emergent (Mis)behavior vs. Complex Software Systems. *EuroSys* (Leuven, Belgium, 18–21 April 2006), pages 293–304. ACM, 2006.

[17] Adam J. Oliner, Ashutosh V. Kulkarni, and Alex Aiken. Using correlated surprise to infer shared influence. *International Conference on Dependable Systems and Networks* (Chicago, IL, 28–30 June 2010), pages 191–200. IEEE/ACM, 2010.

[18] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. *Recovery-oriented computing (ROC): motivation, definition, techniques, and case studies*. Technical report – UCB/CSD–02–1175. UC Berkeley, March 2002.

[19] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of TCP throughput collapse in cluster-based storage systems. *Conference on File and Storage Technologies* (San Jose, CA, 26–29 February 2008), pages 175–188. USENIX Association, 2008.

[20] Raja R. Sambasivan, Alice X. Zheng, Elie Krevat, Spencer Whitman, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R. Ganger. *Diagnosing performance changes by comparing system behaviours*. Technical report 10-107. Carnegie Mellon University, July 2010.

[21] Kai Shen, Christopher Stewart, Chuanpeng Li, and Xin Li. Reference-driven performance anomaly identification. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Seattle, WA, 15–19 June 2009), pages 85–96. ACM, 2009.

[22] Benjamin H. Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. *Dapper, a large-scale distributed systems tracing infrastructure*. Technical report dapper-2010-1. Google, April 2010.

[23] Eno Thereska, Brandon Salmon, John Strunk, Matthew Wachs, Michael Abd-El-Malek, Julio Lopez, and Gregory R. Ganger. Stardust: Tracking activity in a distributed storage system. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Saint-Malo, France, 26–30 June 2006), pages 3–14. ACM, 2006.

[24] Avishay Traeger, Ivan Deras, and Erez Zadok. DARC: Dynamic analysis of root causes of latency distributions. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Annapolis, MD, 02–06 June 2008). ACM, 2008.

[25] Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger. Argon: performance insulation for shared storage servers. *Conference on File and Storage Technologies* (San Jose, CA, 13–16 February 2007), 2007.

[26] Adam Wierman and Mor Harchol-Balter. Classifying scheduling policies with respect to higher moments of conditional response time. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Alberta, Canada, 06–10 June 2005), pages 229–240. ACM, 2005.