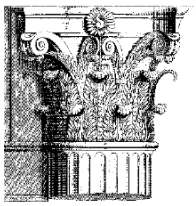
Traveling to Rome: a retrospective on the journey



john wilkes
HP Laboratories, Palo Alto, CA
john@e-wilkes.com

Abstract

Starting in 1994/5, the Storage Systems Program at HP Labs embarked on a decade-long journey to automate the management of enterprise storage systems by means of a technique we initially called attribute-managed storage. The key idea was to provide declarative specifications of workloads and their needs, and of storage devices and their capabilities, and to automate the mapping of one to the other. One of many outcomes of the project was a specification language we called Rome[1] – hence the title of this paper, which offers a short retrospective on the approach and some of the lessons we learned along the way.

Categories and Subject Descriptors D.4.2 Storage Management, D.4.5 Reliability, D.4.8 Performance, I.6.5 Model Development, K.4.3 [Organizational Impacts] automation, K.6.2 Installation Management, K.6.4 System Management.

General Terms Algorithms, Management, Measurement, Performance, Design, Economics, Reliability, Experimentation.

Keywords storage management; attribute-based storage; declarative system management; storage performance models; solvers.

1. Before the beginning

In the late 1980s, I had worked on a scalable storage system called DataMesh [Wilkes1989], which advocated (about a decade too soon!) building a storage system out of intelligent building blocks containing a disk drive, some local processing power, and a high-speed network port. The idea was to connect these together into a mesh, and build a storage system that could be scaled to meet whatever performance or availability demands were placed on it. It quickly became obvious that such a beast would be a nightmare to control and configure if viewed a disk at a time, so we started to think about how to delegate control of design choices to it, starting with failure recovery goals [Wilkes1990].

DataMesh never took off. But the seed of an interesting idea had been planted.

2. Setting out

In 1994, the team I was then part of was finishing up helping our colleagues on the HP AutoRAID disk array project [Wilkes1996]. AutoRAID automated the process of migrating stored data between mirrored and RAID 5 storage tiers, taking account of access patterns, available space, and reliability goals – completely transparently to its users. We asked ourselves, "What if we could apply the AutoRAID ideas to an enterprise-scale storage system that spanned multiple disk arrays?" That is: what if users of large-scale storage systems didn't have to micro-manage the data placement, choice of RAID level, and kind and number of storage devices to purchase? What if the system could work these things out for itself, given a specification of what the customer wanted? The obvious motivations applied: reduced system management costs; lower-cost system designs, faster (and more accurate) response to changing inputs; and fewer errors injected, because there would be less need for human intervention.

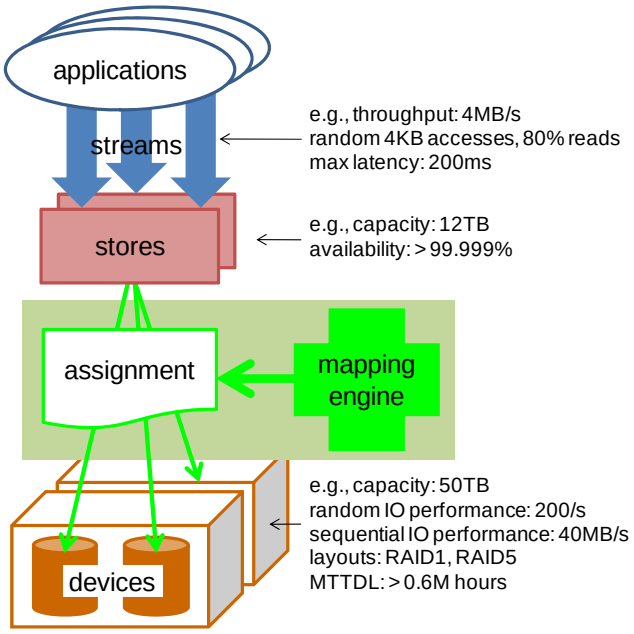


Figure 1. The mapping problem for an attribute-managed storage system.

To accomplish this, we chose to separate the specification of what was desired from the process used to get to an answer – i.e., a

[1] The code names chosen by the HPL Storage Systems program team for the various project components were derived from an architectural theme consistent with our logo – a Corinthian column. Over time, this progressed towards names with a generally classical bent. We apologize for none of them!

declarative system for storage management. The name we chose was *attribute-managed storage* [Golding1995], by comparison to IBM's system-managed storage [Gelb1989]. The overall structure of the problem we tackled is shown in Figure 1.

Starting from the top, one or more applications generate access *streams* that are directed towards one or more *stores*, or storage containers. Attributes associated with each stream capture the dynamic aspects of the workload: the rate at which data is accessed, whether it is read or written (or both), a desired maximum latency, and properties of the access pattern, such as whether it is largely sequential, or random-access, the sizes of requests, their burstiness, correlations between these aspects, and so on. Attributes associated with stores capture the static aspects of the containers, such as how much data they contain, and their desired availability. Finally, stores are mapped onto *devices* – real containers, such as disk drives and disk array logical units, which have attributes that capture their capabilities – capacity, performance, reliability (MTTDL, or mean time to data loss), cache behavior, and so on.

We called the process of assigning stores to devices the *mapping problem*, and proposed to solve it automatically.

Different aspects of the mapping problem included "how many devices are needed to support this load?"; "how much load can this set of devices support?"; and "half my data center has just burned down – which subset of the load can I still support?" Expressing choices between different applications or portions of the load caused us to start thinking about utility, although we elided this complication for much of our early work.

In practice, we spent the majority of our time focused on the first question, on the grounds that most users had a set of work they wanted to get done, and were interested in seeing how to support it. Designing for green-field sites that used only new resources was plenty hard enough, we felt. In retrospect, we somewhat under-estimated the importance of deploying designs into existing environments.

3. Packing for the journey

We began by generating a mathematical formulation of the mapping problem as a constraint-based optimization problem, with the constraints being things like "all workloads should be assigned exactly once", and "no capacity limit should be exceeded" (which covered both storage space and storage device utilization), and with objective functions of the form "minimize the cost of a complete solution" or "maximize the utility" [Shriver1996]. In practice, the majority of our work focused on designing storage systems to meet a particular performance goal while minimizing the overall system cost.

Two outcomes were observable at this stage: a first, clear specification of a set of parameters and attributes for workloads, stores, and storage devices; and the need for models to determine whether constraints were satisfied.

Adding up storage capacity to check a capacity constraint is trivial; determining if the load imposed by placing a set of stores on a device would be too high is much trickier. We quickly ruled out simulations as being too costly, because the "does it fit?" question needed to be asked many, many times in the inner loop of the assignment engine. To provide the necessary efficiency, we adopted analytic models for the expected behavior. Our background in simulation models for storage devices [Ruemmler1994] led us to a set of analytical models for disk devices that was more complete than most, and yet executed quickly [Shriver1998].

We had started down the path of analytical performance models that would occupy us for much of our journey.

To help ground our work, we picked the TPC-D benchmark [TPCD1995] as a representative sample of the kind of application we would have to cope with; we used it as a load generator, not as an audited benchmark. Taking I/O traces of a system running this load showed us that there were several distinct phases in which one portion of the system was heavily used while another lay idle – and vice versa. Time-sharing the storage resources between different phases could save as much as a factor of six in storage system cost. We addressed this issue developing a sophisticated set of performance models that could handle both short-term workload peaks and correlations between longer-term workload behaviors [Borowsky1998].

Somewhere around this time it became clear that our ability to specify attributes and constraints would always exceed our ability to build storage-device models for them!

We used the I/O monitoring technology built into the HP-UX operating system to provide insights into storage system performance, so it was natural for us to build tools that applied a host-based perspective to overall storage behavior, and emphasize the application perspective rather than the storage device one. To explore the data we had, we developed a set of analysis tools – first a trace analysis package called Rubicon, the second a highly-compressed representation and analysis package called DataSeries – since made available as open source [Anderson2009]. The Buttress system allowed time-accurate replay of these traces against a real system [Anderson2004].

So far, we had just been modeling single disk drives. Our real target was disk arrays, which introduced a great many complications in the performance models for various RAID levels [Varki2004]. Hard work on analytical device models eventually addressed these [Uysal2001].

Nonetheless, the time required to generate a set of calibrated storage device models proved troubling – as did the fact that it took a set of highly competent people with PhDs to do it. An alternative approach was needed. We found it in a careful application of brute force. Instead of hand-crafting performance models to predict the likely behavior of a storage device from a priori knowledge about their design, we built models that extrapolated the likely behavior from sets of stored measurements – lots of them. We called this approach table-based modeling [Anderson2001]. Spline-based interpolations to fit the data, and being careful about unwarranted extrapolations, gave us accuracies similar to – or better than – the analytic models, with comparable or better running times, and considerably less work on our part. Even better, the set of measurements could easily be augmented from data obtained from the target system, once it was running, thereby both extending the models and increasing the model's accuracy in exactly the areas where it was most useful.

4. On the road, outbound

Early on, it became clear that the search space we wanted to explore was rather large: this is a variant of the multi-

dimensional, multi-knapsack problem, which is (of course) NP complete – and the scale at which we were operating largely precluded exhaustive search.

We gave the name *solvers* to the tools we used to explore alternative assignments of work to devices. Our first attempt at a solver was called Forum; it handled the single-device models described above, and used greedy hill-climbing to select the best alternative [Borowsky1997]. A few simple heuristics for ordering the examination of alternatives were explored, including (repeated) randomization of the order to consider workloads for assignment; sorting the workloads on various attributes, and using the Toyoda algorithm for deciding which device to pack the next load onto [Toyoda1975].

Forum tackled performance for single storage devices. A completely separate tool, Corbel, was the first to tackle the conjoint design problem for availability and performance at the same time [Amiri1996]. Corbel synthesized RAID designs, tested their availability against the objectives associated with stores, and then selected a suitable design from the ones that were left. Unfortunately, Corbel was never integrated into our mainstream code base – partly because it relied on a somewhat hard-to-use Markov chain analysis tool that was written in Fortran. Corbel used a greedy first-pass assignment process that took the raw hardware cost plus the cost of downtime into account, followed by a refinement pass that fixed up the solution by selective moving of a few stores that were not well-matched to their placements.

5. Making good progress

Our second attempt to support disk arrays was a solver called Minerva [Alvarez2001]. It tackled the problem in two parts: first, it tagged workloads with the recommended kind of RAID level that they should be assigned to, and then it performed a Forum-like assignment using the RAID type as an additional constraint in the matching step. As with Corbel, a final optimization pass cleaned up a few stragglers – especially stores that ended up consuming an entire RAID group to save cost.

For Minerva to work well, we had to make good choices about deciding which RAID level to use for each store [Anderson2002]. Using simple rules of thumb – as a human might do – produced acceptable answers, but integrating the choice into the process of assigning stores to devices did much better, albeit at the cost of additional computation time.

At one point we thought that genetic algorithms (GAs) seemed like an obvious approach to this problem: the species genotypes would represent the current sets of assignments of load to devices, and mutations and combinations would explore the space of alternatives in an efficient fashion. Having tried the experiment, we learned that the cost of evaluating each of the solutions was so high that it dominated the running time of the GA solver, even after aggressive memoization to avoid recalculating previously-encountered partial results. We moved on.

Initially we had been leery of trying to do performance- and availability-based assignments simultaneously because of the huge search space that the assignment problem engenders. However, Eric Anderson was able to get around this problem by constructing a solver that used speculative exploration and back-tracking using a tree-like representation of the design of a storage system and its assignments. The solver, called Ergastulum,[2] performed much faster than Minerva, and was able to explore a great many more alternatives [Anderson2005].

6. Arriving at the destination

The material so far has described how we developed solutions to the declarative design of a single storage system configuration. But our goal was always to develop a way to make the storage system self-managing – by which we meant self-configuring, self-optimizing, self-healing, and all of the other self-* objectives.

The approach was straightforward – at least in principle: (1) take a specification for what is wanted; (2) build a storage system that matches those needs; (3) deploy the application or workload on that system; (4) monitor it to see if it is meeting the actual needs of the workload; (5) re-design if necessary, and (6) reconfigure the storage and migrate the application's data – preferably while the application is still running. Figure 2 shows the idea.

Hippodrome was the name of the system we devised to do all this [Anderson2002a]. It went one better: it didn't need a detailed specification of the performance requirements of the workload, just the capacity and availability needs. It would run the application, measure the result, design and deploy a system to meet those needs, and iterate until the result stabilized – typically in only 2 to 3 iterations. This was the system we had been aiming for all along. We liked to describe the way it would optimize the design of a storage system over a weekend, with no human intervention.

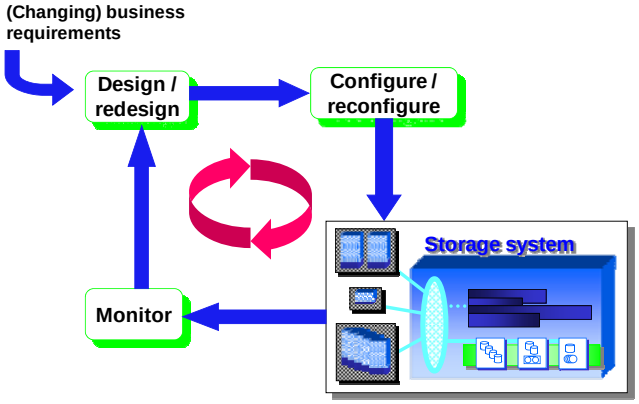


Figure 2. The design, configure, run, monitor loop.

7. Language barriers

Workload descriptions (streams plus stores), device capabilities, models, objective functions, and configuration settings for our tools all needed writing down. In previous work we had invented a way of using Tcl to configure a complicated simulation system [Golding1994], and we used the same approach to express the solver's inputs and outputs, allowing us to feed the results from one run into another. Our language, which we christened Rome,

[2] The name means a private prison or workhouse for slaves attached to a Roman farm. It was selected when Eric was a summer intern in our group – he claims that it seemed like a good idea at the time. Regrettably, the ACM TOCS reviewers took aversion to it, and we had to drop it from the published version.

proved to be a convenient, flexible way to record attributes and other specifications; it readily supported nesting of components and dynamic extensibility. Its interpreted nature made it easy to ignore elements that were not understood, providing support for simple versioning.

Rome stood us in good stead for quite some time, but eventually became encrusted with hidden assumptions about the meanings of various elements and their relationships – the semantics proved ambiguous between different tools.

Rome 2 was an attempt to provide a clean specification for both the syntax and semantics of the language we were using. It was derived from the *de facto* version, and followed it quite closely in many ways. We should have done this sooner; by the time Rome 2 was ready, it was too late – the team had moved on to other goals, and it was never adopted.

Towards the end of this effort, modeling systems like the Common Information Model (CIM) from the Distributed Management Task Force (DMTF) were coming to prominence, but we chose not to switch to them – partly because we found the representations somewhat unnatural and unwieldy; partly because the attributes that had been defined by then didn't yet handle the things we needed; and partly because our tools were already "good enough" for our research focus.

Another lesson we learned was the importance of separating the semantics of a language from its encoding. Well-meaning people kept on pressing us (unhelpfully) to use XML – as if that would solve any of our problems. To hammer this point home, two forms of representation were provided for the Rome language: the native version (derived from the Tcl syntax) was called Latin; the alternative XML one was called Greek – and was typically 2–3 times as long and essentially unreadable. In practice, having a language that humans can manipulate, plus automated translations back and forth into a more "standard" representation like XML, is the right way to proceed – a lesson that has yet to be learned by many groups, it seems!

8. The journey back

One slightly troubling aspect of our approach that we had chosen to elide was how we were going to answer the question: "where do the requirements come from?" Hippodrome offered one way out (measure them), but that doesn't work for systems that don't exist yet, or for non-measurable metrics such as availability or reliability targets.

Simply specifying a target availability level is fine in some circumstances, but is inadequate when a storage system can tolerate partial failures, which often result in data remaining accessible, but at reduced performance. The notion of *performability* captures is better: it captures the fraction of the time a system is available at what performance – including no performance at all [Alvarez2001a]. Sadly, people find it hard enough to articulate availability goals, let alone performability ones. Nonetheless, it is still a useful way of capturing the availability achieved by a system, and forms the basis for many service-level agreements.

We never came up with a much better answer for obtaining performance objectives, but we did make some headway on the robustness requirements. Rather than asking for target objectives at a technical level, we realized that it would be easier to capture business needs in financial terms such as the hourly penalty rate associated with an outage (unavailability) or data loss (rollback) – and that these are the real drivers for many decisions anyway.

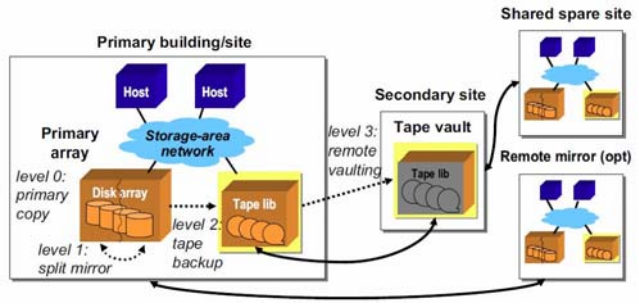


Figure 3. A sample multi-site disaster-tolerant storage system design.

Casting both penalties and system costs in financial terms allowed us to treat designing a system's availability, reliability, or performability properties an optimization problem, with the goal of minimizing the sum of the expected penalties and the cost of achieving a particular outcome. Quite complicated trade-offs can be handled automatically with this approach, such as the choices indicated in Figure 3, which shows a fairly typical situation, involving several sites, local and remote mirrors, and both disk- and tape-based recovery techniques.

We initially applied this approach to designing the storage system itself [Keeton2004], then to evaluating how well the storage system behaves when things go wrong [Keeton2004a], and finally to automating the design of the recovery process once things have started to go wrong [Keeton2006], reducing the likelihood that people will exacerbate the situation by making a bad decision when under stress.

9. Entertaining excursions

Hippodrome required the ability to reconfigure a storage system between iterations, but there are plenty of other reasons to want to move (or migrate) data from one setup to another.

The setup here is simple: descriptions are provided of an initial data layout and a final one, and the goal is to derive a plan that migrates the data from one to the other, while minimizing some goals, such as the number of spare staging areas needed, or the elapsed time, or both. Constraints might include the rate at which data can be moved, the amount of spare capacity, or the amount of parallelism allowed. We found that applying a declarative problem-specification plus an automated solver to the migration problem led to similar dividends as with the assignment problem [Saia2001, Anderson2008]. The problem is by no means completely solved: our work didn't support changing the format of containers or taking performance effects such as network bottlenecks into account.

Our early work ignored the storage area network (SAN) that is typically used in an enterprise to connect host computers to their storage devices. This was obviously an over-simplification, although it served us well for a long time because SAN costs are often only a small fraction of the total system cost. Nonetheless, we became aware of SAN designs that cost several million dollars to build, and decided this was worth some attention.

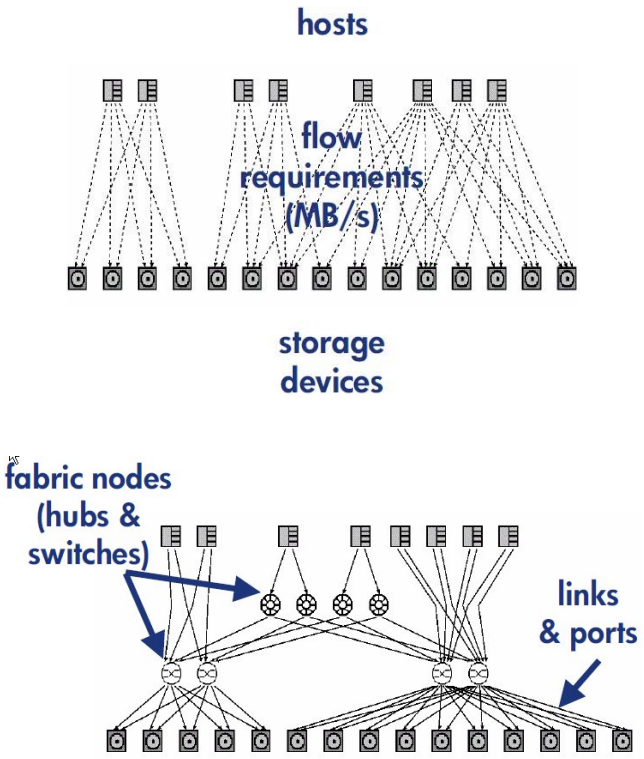


Figure 4. A sample SAN design problem (top: input, bottom: output).

The approach we took was to treat it as another design problem. The inputs were the output from one of our storage system design tools, together with the objective function – typically minimize cost, perhaps with an requirement that there be no single point of failure to increase availability. The output was a design for the SAN, including the choice of devices (FibreChannel switches and hubs), the network topology to use, and the wiring diagram. Leaving the topology as a free variable gave us the freedom to design solutions that were often significantly cheaper than the regular structures of the kinds humans preferred, albeit at the cost of symmetry [Ward2002]. Figure 4 shows a sample problem, with a solution that avoids any single point of failure.

As described so far, the storage system design cycle is a long-lived one, operating at the timescale of provisioning decisions (hours or days). In order to cope with shorter-term fluctuations, it's necessary to provide a finer-grained control mechanism, some way to enforce quality-of-service goals at runtime [Karlsson2004, Wang2007]. Doing so requires a clear understanding of the objectives that it is intended to enforce: another example of the need for precise specifications.

10. Returning home

What did all this teach us? First: declarative approaches are powerful – they can be made to work at significant scale and complexity, and across a wide range of problems. The capabilities of the technology are exciting; and the use of goal-based declarative specifications seems much cleaner than rule-based or process-based ones such as workflows.

Second: deploying such systems is much more than a technical problem. In fact, I believe that the single greatest barrier to the adoption of automated solutions is not generating the technology to build them, but our ability to persuade the likely users that they should trust that the systems will do the right thing, in all circumstances. To this end, we need to invest more in making our systems trustworthy – which means ensuring that they don't surprise people; making it easier to express what we want them to do; putting limits on what they can do without our consent; and explaining their decisions when requested.

Ultimately, we need to remind ourselves that we are building systems to serve people, and the success of our technical accomplishments will be dictated by how comfortable we can make those people with what we are accomplishing on their behalf. Success will bring many benefits – lower costs, fewer errors, more rapid responses to environmental changes and system faults, and more time for people to spend their lives on creative pursuits, rather than repetitive drudgery. It *will* be worth it!

11. Acknowledgements

The work described here was almost entirely done by others – a talented set of colleagues whom I had the pleasure and good fortune to work with over the last two decades. My thanks to them all.

12. References

Many of these papers can be found online at: http://www.hpl.hp.com/research/ssp/papers.

[Alvarez2001] Guillermo A. Alvarez, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph Becker-Szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, Alistair Veitch, and John Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems* 19(4):483-518, November 2001.

[Alvarez2001a] Guillermo A. Alvarez, Mustafa Uysal, and Arif Merchant. Efficient verification of performability guarantees. *International Workshop on Performability Modeling of Computer and Communication Systems* (*PMCCS 5*, Erlangen, Germany), pp. 95-99, September 2001.

[Amiri1996] Khalil Amiri and John Wilkes. *Automatic design of storage systems to meet availability requirements*. Technical report HPL-SSP-96-17, HP Laboratories, August 1996.

[Anderson2001] Eric Anderson. *Simple table-based modeling of storage devices*. Technical report HPL-SSP-2001-4, HP Laboratories, July 2001.

[Anderson2002] Eric Anderson, Ram Swaminathan, Alistair Veitch, Guillermo A. Alvarez and John Wilkes. Selecting RAID levels for disk arrays. *File and Storage Technology* (*FAST'02*, Monterey, CA) pp. 189-201, January 2002.

[Anderson2002a] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: running circles around storage administration. *File and Storage Technology* (*FAST'02*, Monterey, CA) pp. 175-188, January 2002.

[Anderson2004] Eric Anderson, Mahesh Kallahalla, Mustafa Uysal, Ram Swaminathan. Buttress: a toolkit for flexible and high fidelity I/O benchmarking. *File and Storage Technology (FAST'04, San Francisco, CA)*, pp. 45-58, March-April 2004.

[Anderson2005] Eric Anderson, Susan Spence, Ram Swaminathan, Mahesh Kallahalla, Qian Wang. Quickly finding near-optimal storage designs. *ACM Transactions on Computer Systems* 23(4): 337-374, November 2005.

[Anderson2008] E. Anderson, J. Hartline, M. Hobbs, A. Karlin, J. Saia, R. Swaminathan and J. Wilkes. Algorithms for Data Migration. *Algorithmica* 51, August 2008. DOI 10.1007/s00453-008-9214-y.

[Anderson2009] Eric Anderson, Martin Arlitt, Brad Morrey, and Alistair Veitch. DataSeries: an efficient, flexible data format for structured serial data. *Operating Systems* Review 43(1), January 2009 .

[Borowsky1997] E. Borowsky, R. Golding, A. Merchant, L. Schreier, E.Shriver, M.Spasojevic, and J. Wilkes. Using attribute-managed storage to achieve QoS. *5th Intl. Workshop on Quality of Service* (*IWQoS*, Columbia Univ., New York, NY), June 1997, pp. 199-202.

[Borowsky1998] Elizabeth Borowsky, Richard Golding, Patricia Jacobson, Arif Merchant, Louis Schreier, Mirjana Spasojevic and John Wilkes. Capacity planning with phased workloads. *Workshop on Software and Performance* (*WOSP'98*, Santa Fe, NM), October 1998.

[DMTF-CIM2008] Distributed Management Task Force, Inc. *Common Information Model (CIM) Standards*, 2008. http://www.dmtf.org/standards/cim/.

[Gelb1989] J. P. Gelb. System managed storage. *IBM Systems Journal* 28(1):77–103, 1989.

[Golding1994] Richard Golding, Carl Staelin, Tim Sullivan, John Wilkes. "Tcl cures 98.3% of all known simulation configuration problems" claims astonished researcher! *Tcl Workshop* (New Orleans), May 1994.

[Golding1995] Richard Golding, Elizabeth Shriver, Tim Sullivan, and John Wilkes. Attribute-managed storage. *Workshop on Modeling and Specification of I/O* (San Antonio, TX), 26 Oct. 1995.

[Karlsson2004] Magnus Karlsson, Christos Karamanolis and Xiaoyun Zhu. Triage: performance isolation and differentiation for storage systems. *International Workshop of Quality of Service* (*IWQoS'04*, Montreal, Canada), pp. 67-74, June 2004.

[Keeton2004] Kimberly Keeton, Cipriano Santos, Dirk Beyer, Jeffrey Chase and John Wilkes. Designing for disasters. *File and Storage Technologies* (*FAST'04*, San Francisco, CA), March-April 2004.

[Keeton2004a] Kimberly Keeton and Arif Merchant. A framework for evaluating storage system dependability. *International Conference on Dependable Systems and Networks*, (*DSN'04*, Florence, Italy), June-July 2004.

[Keeton2006] Kimberly Keeton, Dirk Beyer, Ernesto Brau, Arif Merchant, Cipriano Santos and Alex Zhang. On the road to recovery: restoring data after disasters. *European Systems Conference* (*EuroSys'06*, Leuven, Belgium), pp. 235-248, April 2006.

[Ruemmler1994] Chris Ruemmler and John Wilkes. An introduction to disk drive modelling. *IEEE Computer* 27(3):17-28, March 1994.

[Saia2001] Jared Saia, Eric Anderson, Joe Hall, Jason Hartline, Michael Hobbes, Anna Karlin, Ram Swaminathan, and John Wilkes. An experimental study of data migration algorithms. *Algorithm Engineering, the Proceedings of WAE 2001: 5th Workshop on Algorithm Engineering* (BRICS, University of Aarhus, Denmark), August 2001). Published as *Springer-Verlag Lecture Notes in Computer Science* 2141, pp. 145-158, August 2001.

[Shriver1996] Elizabeth Shriver. *A formalization of the attribute mapping problem*. Technical report HPL-SSP-95-10 revision D, HP Laboratories, July 1996.

[Shriver1998] E. Shriver, A. Merchant, and J. Wilkes. An analytical behavior model for disk drives with readahead caches and request reordering. *Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 182–91, June 1998.

[TPCD1995] Transaction Processing Performance Council, TPC-D benchmark, April 1995. http://www.tpc.org/tpcd.

[Toyoda1975] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21(12):1417–27, August 1975.

[Uysal2001] Mustafa Uysal, Guillermo A. Alvarez, and Arif Merchant. A modular, analytical throughput model for modern disk arrays. *9th Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (*MASCOTS'01*, Cincinnati, Ohio), pages 183-192, August 2001.

[Varki2004] Elizabeth Varki, Arif Merchant, Jianzhang Xu and Xiaozhou Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 15(6):559-574, June 2004.

[Wang2007] Yin Wang and Arif Merchant. Proportional share scheduling for distributed storage systems. *File and Storage Technologies* (*FAST'07*, San Jose, CA), February 2007.

[Ward2002] Julie Ward, Michael O'Sullivan, Troy Shahoumian, and John Wilkes. Appia: automatic storage area network design. *File and Storage Technologies* (*FAST'02*, Monterey, CA), pp. 203-217, January 2002.

[Wilkes1989] John Wilkes. *DataMesh --- scope and objectives.* Technical report HPL-DSD-89-37rev1, HP Laboratories, July 1989.

[Wilkes1990] John Wilkes and Raymie Stata. Specifying data availability in multi-device file systems. *4th ACM-SIGOPS European Workshop* (Bologna, Italy), September 1990, published as *Operating Systems Review* 25(1):56-59, January 1991.

[Wilkes1996] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems* 14 (1):108-136, February 1996.