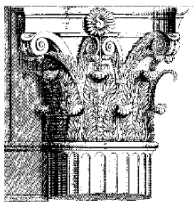
Traveling to Rome: a retrospective on the journey



john wilkes

HP Laboratories, Palo Alto, CA
john.wilkes@hp.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'd 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 you might 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, we were about to finish helping our colleagues on the HP AutoRAID project [Wilkes1996] and asked ourselves – "what if we could apply the same principles to an enterprise-scale storage system?" 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 were offered: reduced system management costs; lower-cost system designs, faster (and more accurate) response to changing inputs; and fewer errors injected.

We wanted to separate the specification of what was desired from the process used to get to an answer – i.e., we were defining a declarative system for storage management. The name we chose was *attribute-managed storage* [Golding1995], by comparison to IBM's system-managed storage [Gelb1989]. Stores (data objects), streams (access patterns), and storage devices were each given attributes that specified their properties or behaviors. Streams were associated with stores. We called the process of assigning stores to devices the *mapping problem*, and proposed to solve it automatically.

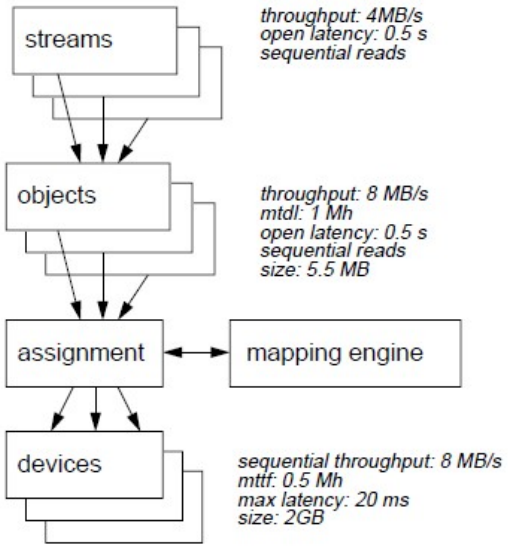


Figure 1. The attribute mapping problem.

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?" (which introduced a need for an early version of utility); and "half my data center has just burned down – which subset of the load can I still support?" In practice, we spent the majority of our time focused on the first question, on the

[1] The 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!

Presented at the "The Rise and Rise of the Declarative Datacentre" research meeting,
Microsoft Cambridge, England, 12-13 May 2008. http://research.microsoft.com/riseandrise/
Published as Microsoft Research technical report MSR-TR-2008-61, pages 49-52.

grounds that most users had a set of work they wanted to get done, and were interested in seeing how to support it.

3. Packing for the outward journey

It was pretty straightforward to generate 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", and with objective functions of the form " minimize the cost of a complete solution", or "maximize the utility" [Shriver1996].

Two additional 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 constraint was trivial; determining if the load imposed by placing a set of stores on a device would be too high was much trickier. We quickly ruled out simulations as being too costly, and resorted to simple 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 this leg of our journey.

To help ground our work, we picked the TPC-D benchmark as a representative sample of the kinds of application we would have to cope with. Taking I/O traces of a (non-audited!) 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. Addressing this issue resulted in us developing a sophisticated set of performance models that could handle both short-term workload peaks and correlations between longer-term workload behaviors [Borowsky1998].

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 clever application of brute force. Instead of carefully crafting models that predicted the likely behavior of a storage device, we built models that extrapolated the likely behavior from sets of stored measurements – lots of them. We called this approach table-based modeling [Anderson2001]; using 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 runtimes, and considerably less work on our part.

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 called the tools we used to explore alternative assignments of work to devices *solvers*. 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 greed 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]. That tackled the problem in two parts: it first tagged workloads with the recommended kind of RAID level that they should be assigned to, and then performed a Forum-like assignment. As with Corbel, a final optimization pass cleaned up a few stragglers – especially stores that ended up consuming an entire RAID group.

To make Minerva 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 some 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.

Initially we had been leery of trying to do performance- and availability-based assignments simultaneously because of the huge search space that it engenders. However, Eric Anderson was able to get around this problem by constructing a solver that used speculative exploration plus 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

[2] The name means a private prison attached to most Roman farms, where the slaves were made to work in chains. 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.

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 migrate the application to the new configuration – preferably while it is still running.

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 2-3 iterations. This was the system we had been aiming for all along.

7. Language barriers

Workload descriptions (streams plus stores), device capabilities, models, objective functions, and configuration settings for our tools all needed writing down. Some years before we had invented a way of marrying Tcl with a complicated simulation system [Golding1994], and we continued this approach as we developed ways to write down these various inputs. The result was a fairly flexible language for recording attributes and other specifications that naturally supported nesting of components and dynamic extensibility (by being interpreted, and making it easy to ignore elements that were not understood).

We christened this language Rome; it 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.

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 may 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. Another lesson was the importance of separating the semantics of a language from its expression. Well-meaning people kept on pressing us (unhelpfully) to use XML – as if that would solve any of our problems. 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 relearned by many groups, I fear.[3]

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.

We never did come up with a better answer for the first problem, but we did make some headway on the latter, by taking a step back and realizing that availability and reliability requirements are ultimately driven by business needs. If we could extract those business needs, we reasoned, we could use them to drive decision-making about the right availability levels to push for.

In fact, we ended up realizing that we could go one better: if business objectives can be expressed in monetary terms – such as the hourly cost of an outage (unavailability) or data loss – we could add that to the [calculated] cost of achieving a particular level of availability or reliability, and treat the result as an optimization problem, with the objective of minimizing their sum. This turned out to work well; first for designing the storage system itself [Keeton2004], for evaluating how well the storage system will behave when things go wrong [Keeton2004a], and – best of all – for working out how to recover once things have started to go wrong [Keeton2006]. We suspect that the latter is particularly valuable, as the likelihood of making errors increases greatly when people are under stress.

9. Entertaining excursions

Hippodrome requires the ability to reconfigure a storage system between iterations, but there are plenty of other reasons to want to move data from one setup to another. We found that applying the same kind of declarative problem-specification plus an automated solver to the migration problem led to similar dividends. 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 moves the data from one to the other, while minimizing the number of spare staging areas needed, or the elapsed time, or both [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.

As described, 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. One approach to this is to enforce quality of service at runtime [Karlsson2004, Wang2007]. Doing so requires a clear understanding of the specifications that it is intended to enforce – another example of the need for precise declarative specifications.

10. Returning home

What has all this taught us about declarative approaches? First: 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: that deploying such systems is much more than a technical problem. In fact, I believe that the single greatest barrier to adoption of such systems is not our ability to generate the technology to build such systems, but our ability to persuade the likely users that they should trust that the systems will do the right thing. 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.

[3] To press 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.

Acknowledgments

The work described here was actually done by a talented pool of colleagues who I had the pleasure – and good luck – to work with over the last two decades. Space precludes listing them, but my thanks to them all!

References

[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.

[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.

[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*, to appear, 2008

[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.

[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* (*EuroSy'06s*, 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.

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

[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.

[Wilkes1989] John Wilkes. *DataMesh --- scope and objectives*. Technical report HPL-DSD-89-37rev1, HP Laboratories, July 1989. http://www.hpl.hp.com/research/ssp/papers/#DataMesh

[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.

[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.