Automatic design of dependable data storage systems

Kimberly Keeton and John Wilkes

Storage Systems Department
Hewlett-Packard Laboratories, Palo Alto, CA
{kkeeton, wilkes}@hpl.hp.com

ABSTRACT

Constructing dependable storage systems is difficult, because there are many techniques to pick from that interact in often unforeseen ways. The resulting storage systems are often either over-provisioned, or provide inadequate protection, or both. We assert that automating our way out of this dilemma is both desirable and achievable, and we present some lessons we have learned from our initial efforts at doing so. The result is a first step down the path of self-managing, dependability-aware storage systems, including a better understanding of the problem space and its tradeoffs, and a number of insights that we believe will be helpful to others.

1. INTRODUCTION

Facts do not cease to exist because they are ignored.
– Aldous Huxley

Hardware breaks. Software has defects. Viruses propagate. Buildings catch fire. Power outages happen. People make mistakes. Much as we might prefer that these events weren't so prevalent, it is only prudent to make plans for their occurrence. In the storage domain, this means making redundant copies of data, isolating these copies as necessary from sources of further problems, and trading off the desire for rapid recovery and minimum data loss against the monetary, operational, and performance costs of achieving them.

Data is the primary asset of most corporations in the information age, and access to that data is vital if businesses are to continue operation. In a 2001 survey [7], fully a quarter of its respondents estimated data access loss costs as more than $250,000/hour – and 8% estimated them as more than $1M/hour. The price of data loss is even higher. Gartner estimates that "two out of five enterprises that experience a [site] disaster … go out of business within five years [16]." Robust, dependable systems are needed to avoid such problems.

We use the term *data dependability* to cover both data *reliability* (i.e., the lack of data loss or corruption) and data *availability* (i.e., that access is always possible when it's desired). Over time, the list of mechanisms that are available to achieve storage system dependability has grown. New ones continue to be invented, and old ones become more widely applicable as the cost of raw storage capacity continues to plummet. The result is a plethora of approaches, techniques, and configuration choices. Each technique provides some of the necessary protection; combined, they can cover a much broader range – but sometimes they can interact in surprising ways.

This design space might be manageable if there were clearly specified, agreed-upon objectives for data dependability. Unfortunately, this is rarely the case, because the large variations in cost, performance, and dependability that are achievable make the tradeoffs hard for human operators and system designers to reason about. For example, humans are known to be poor estimators of low-risk events with large effects.

As a result, design choices are often made in an ad hoc manner, focusing more on the practical issues of setting the configuration knobs for a chosen technique than on trying to achieve a desired dependability level. Administrators may not even understand the data dependability ramifications of their design choices. As a result, the systems deployed are all too often not as dependable as they need to be, or more costly, or both.

We believe that the exploration of the design space and its tradeoffs can usefully be automated. Our eventual goal is a storage system that is completely self-managing, but our initial steps are concentrated on the interactive design of systems. This requires a rapid trial-response cycle, measured in seconds, not hours or days. In turn, this requires that the design-space exploration be automated, and results be presented in ways that are helpful to storage system designers, rather than the developers of storage mechanisms.

Contributions

The first contribution of this paper is a summary of dependability-oriented design options and fault types that storage system designers are trying to tackle. Our message is that the space is a rich one, and not as simple as it might first appear. That richness contributes to the difficulties that people have in understanding and meeting their goals.

The second contribution is a description of a simple tool that successfully attacks a portion of the problem, along with some initial results. Although it is only an exploratory venture, we have already found it helpful in formalizing our understanding of the problem, and in demonstrating promise in the approach.

The third contribution is a set of lessons learned so far. We conclude with a summary of next steps.

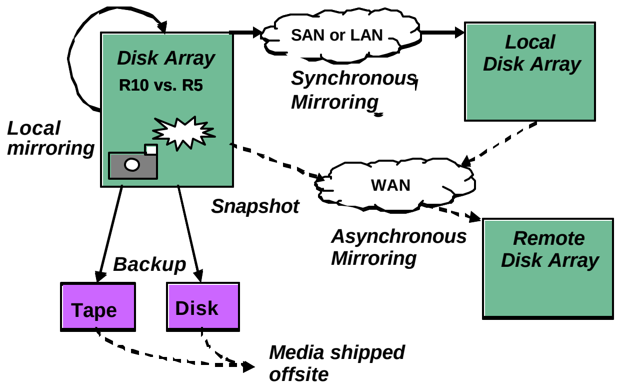

Figure 1: data protection techniques handled by our initial dependability-design tool.

2. THE PROBLEM SPACE

The chief cause of problems is solutions. – Anonymous

The storage system design space is large, thanks to the multitude of different failure types that it is desirable to protect against, and the proliferation of data protection techniques. Without attempting an exhaustive enumeration of either, we offer a brief taxonomy of the two to communicate the richness of the space that they represent.

Sample failure types

- *Storage hardware (container) failures* – disk drive, disk array, file server, rack, building, site (e.g., fire), region (e.g., flood or earthquake)
- *Common shared infrastructure failures* – power, air conditioning
- *Software-induced data corruption* - storage device firmware, networking components, host operating system, application defects, virus
- *Human/operator/user error* – accidental mis-configuration, deletion or overwriting, malicious insider attacks (e.g., disgruntled employee), external attacks (e.g., denial of service)

Different installations may care more about different failure types than others. For example, reasonably well-conditioned power is common in developed regions, but cannot be taken for granted in many other parts of the world.

Data protection techniques (a partial list)

- *Inside one storage device* (e.g., a disk array): this includes both partial redundancy (e.g., RAID5, copy-on-write snapshots, file system metadata journals) and full redundancy (e.g., RAID1, RAID10, database logs).
- *Between storage devices*: this includes the cross product of full redundancy (e.g., mirroring) and partial redundancy (e.g., erasure codes, RAID-5) at the same level of the storage hierarchy (e.g., array to array) or across different ones (e.g., array to tape, or array to slow, cheap disk storage).
- *Update frequency*: synchronous, lock-step with foreground updates; continuous asynchronous (e.g., remote mirroring, logging); and batched (e.g., nightly backup). See [9] for a more complete classification for remote mirroring.

Combinations of techniques are also important. One popular combination includes local RAID-5, remote mirroring, snapshot and backup to tape. RAID-5 provides protection against disk failures, remote mirroring guards against site failures, snapshots address user errors, and tape backup protects against software errors and provides archival.

After a failure, fresh resources are often required for recovery. The best case occurs when these are already acquired, installed, configured, formatted, and immediately available for allocation to the restoration process (i.e., not in use for anything else). Removing any one of these properties can introduce delay – the worst case being when new equipment has to be purchased, delivered, installed, connected, configured, tested, and made available. In practice, a wide range of "somewhat ready" equipment sources exist, including dedicated hot spares, unused local resources, remote dedicated or shared disaster-recovery facilities (which can be owned or leased), all the way out to the equipment manufacturer's delivery pools. Gaining access to physically-available equipment may require negotiation or the migration of the current workload (e.g., for a shared facility).

In addition to impacting the delay until the resources are available, a secondary effect of these variations is to complicate the accounting model of a potential solution's cost.

Tradeoffs

Different protection techniques offer a range of different tradeoffs:

- *Protection overhead* – including capacity, performance impact on the foreground workload, and downtime during which applications must be quiesced.
- *Recovery time* – how long does it take to get the data back? (Note that data recovery time is only a portion of the outage duration experienced by application users.)
- *Recovery points* – how much (and which) data can be returned to its pre-loss state?
- *Cost* – what are the direct and indirect monetary costs for a particular solution? Direct costs include equipment purchase or rental costs, human administrator salaries, and lost revenues from degraded mode operation. Indirect costs include lost worker productivity, user dissatisfaction and damaged corporate reputation.

For example, mirroring offers rapid recovery after a loss of one copy, but has 100% capacity overhead and slower write times, and still provides no protection against accidental data deletion.

Costs are important: we know of several cases where the initial desire to "protect everything" evaporated after the financial costs of doing so (redundant storage systems, multiple sites, repeated operator training and trial runs, etc.) became apparent. Brute-force over-provisioning is not usually sensible at enterprise scale, although it can be quite effective for smaller systems.

To reduce overall costs, some companies have chosen to implement systems that satisfy only a portion of their data dependability needs, investing in an insurance policy to cover the remaining potential losses. Others deliberately choose to forgo protection, and live with the risk.

Another approach to reducing costs is to use multiple explicit data dependability levels for different subsets of the data. There's no need to pay for the most expensive alternative for information that is ephemeral, such as a temporary file, or can be reconstructed easily, such as a database index. This approach allows scarce resources to be focused on the most important, or most valuable, data.

In summary, we have found that the problem space is large, and the design space is rich. The potentially devastating consequences of failures, coupled with their relative rarity, means that traditional trial-and-error solutions are simply inappropriate. Something more stringent is needed if we are to make dependable systems the norm, rather than a pleasant exception.

3. AUTOMATING DATA DEPENDABILITY DESIGN

We are continually faced by great opportunities brilliantly disguised as insoluble problems. – Lee Iacocca

We believe that the key challenge is that of *designing* highly dependable storage systems – automatically. By design, we mean the problem of choosing which data protection techniques to apply to which data, and of setting the corresponding configuration knobs appropriately. To that end, we have been exploring techniques for automating data dependability design.

Design is a fundamental process, both for developing an initial system, and for adapting a system to changes in its load, goals, or environment. Thus, even though we are presenting results from a standalone design tool here, our eventual goal is to include something like it in a dynamic, self-managing system.

In order to accelerate our understanding of the problem, we began with a deliberately narrow focus: handling the combination of (1) remote mirroring, (2) local mirroring, (3) tape backup, and (4) site failover for disk arrays (see Figure 1). Despite this, we found we had to put together a disturbingly large number of design choices, and cope with a wide variety of configuration choices. For example, in addition to the dimensions outlined above, we had to think about:

- the required bandwidth to mirror data, and the number (and cost) of the network links between the two sites
- how often to take full backups and how often to make incremental ones, the bandwidth required to make backups, and the number of required tape devices
- when to move backup tapes off-site for disaster recovery
- whether to rebuild a stricken site, or build a new one somewhere else

The basic approach we took was to determine a minimum-cost solution, by trying a number of candidate designs.

Cost: we measured cost with two components: *outlays*, such as the purchase price or capital depreciation for storage equipment, operating costs, lease rates; and *penalties*, which are the cost of data loss and data inaccessibility, expressed as $/hour and $/byte, respectively. We allowed different data loss costs for "recently written" data that might still be buffered, and for "stable" data.

Workload: we used a simple workload model, which included capacity, write rate and its burstiness, and the write working set size, derived from measurements of our local 1.36TB workgroup file server. This model drove several performance-sensitive design choices, such as the number of remote mirroring links needed.

Performance: we deliberately chose to ignore *all* other performance issues.

To expedite experimentation, we turned to a pre-existing linear optimization package that could handle value- and integer-value constraints, inter-dependencies, and already had a decent user interface: the solver package available in Microsoft Excel [12]. This proved to be a useful tool for early experiments, and let us get feedback from an early working prototype in only a couple of days. The results presented in Section 4 are from this implementation.

4. RESULTS

Our initial results explore the use of a single data protection technique at a time to survive the failure of a disk array for a file server workload. As shown in Figure 1, we considered several alternate techniques, including tape backups, remote mirroring and local mirroring. We present a few representative results here.

Our design parameters varied across the following ranges: 1–16 remote mirroring links, 10 seconds to 24 hours duration for asynchronous batches [9], 1–16 tape drives, 4–48 hour tape backup window, 0–27 incremental backups between each full backup pair, and 1 minute to 34 hours time to prepare spare resources. We fixed penalties as follows: stable data loss $20k/MB, recent data loss $2k/MB, data inaccessibility $50k/hour. Disk array costs were calculated based on the list prices for HP's mid-range EVA (Extended Virtual Array). Tape system costs were calculated based on list prices for HP's ESL9595 tape library. We used a 3-year capital depreciation period.

Figure 2 illustrates the recovery times experienced by these design alternatives after an array failure. In this model, we assume that recovery is completed when the data has been reconstructed onto a new array at the local site. The line labeled "Default (no spare)" includes weekly full backups with daily incremental backups and asynchronous mirroring with a 10-second update batching window, both with no spare resources.

Perhaps surprisingly, the recovery time for the default remote mirroring designs is considerably higher than the recovery time for the default tape backup or local mirroring. The reason is that reloading from fast tape drives is faster than retrieving all the data from the remote site across the WAN links, which had been provisioned to support the remote mirror update traffic, not the restore traffic. We observed an analogous scenario with the number of tape drives used in the backup case: fast recovery may need more tape drives than are required to create the backup during the requested backup window.

The line labeled "Min RT (no spare)" illustrates the recovery times when the wide area links and tape drives are provisioned to *min*imize *r*ecovery *t*ime. Despite this, all of the "no spare" designs require 35 to 40 hours for recovery, because of the time required to order, set up and configure a replacement disk array. Having a hot spare array available and appropriately configured can reduce this time to less than five hours.

Table 1 describes the data loss experienced by our design alternatives under array failure. In the case where no additional dependability design is applied to the primary array, the array experiences a complete data loss (over a TB for this example). Tape backup limits the window of vulnerability to recent data loss, based on the frequency of backups. The default daily backup scheme provides a worst-case data loss of two-days' worth of recent data. This loss potential can be reduced by conducting backups more frequently and over a shorter backup window, as shown in the figure. As expected, the mirroring solutions experience nearly zero data loss.

The stable data loss for our dependability designs is zero because we make the simplifying assumption that these protection techniques will operate correctly (e.g., tape drives won't break, tape media won't become corrupted or unreadable, inter-array links won't become inoperable). If we relaxed this assumption, the stable data loss values would increase.

Figures 3 and 4 quantify the outlay and penalty costs associated with our default and optimal dependability designs (note the log scale on the y axes). Outlay costs are provided on an annualized basis, and penalty costs are provided on a per-failure event basis. As expected, the outlay cost for tape backup requires the smallest increment over the primary array, followed by the local mirroring options, and finally the remote mirroring options.

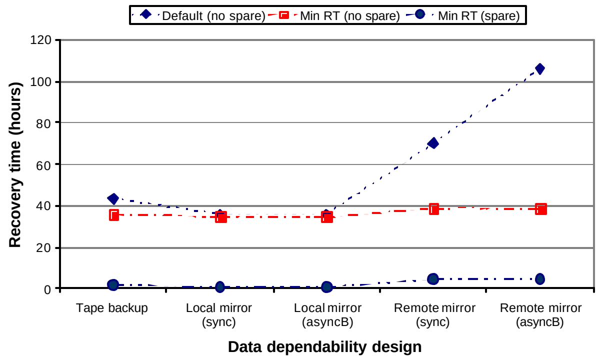

Figure 2: recovery time behavior.

Table 1: data loss behavior.

Data protection technique	*Stable data loss*	*Recent data loss (default)*	*Recent data loss (min DL)*
Primary array only	1360 GB	0 GB	0 GB
Tape backup	0 GB	5 GB	1 GB
Local mirror (sync)	0 GB	0 GB	0 GB
Local mirror (asyncB)	0 GB	14 KB	14 KB
Remote mirror (sync)	0 GB	0 GB	0 GB
Remote mirror (asyncB)	0 GB	14 KB	14 KB

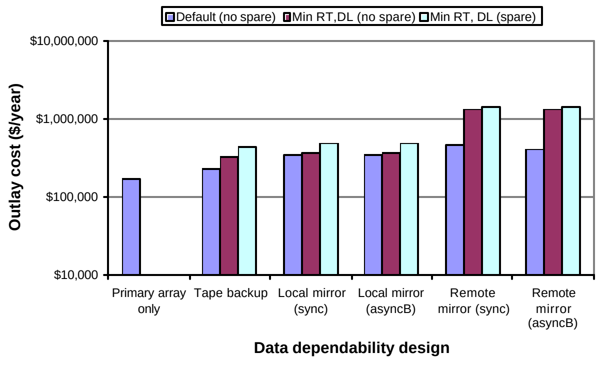

Figure 3: outlay costs for "typical" and "best" designs.

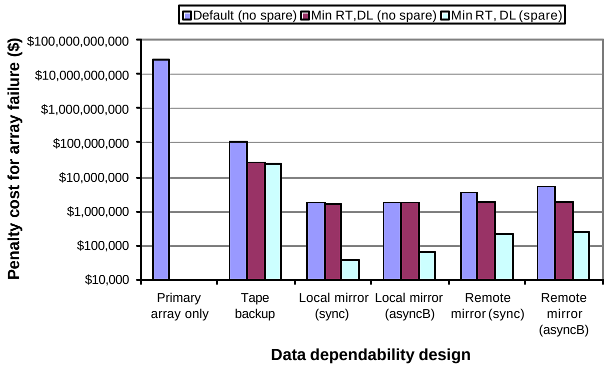

Figure 4: penalty costs for "typical" and "best" designs.

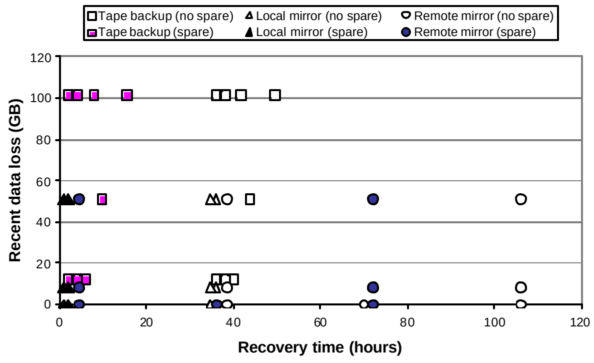

Figure 5: recent data loss vs. recovery time tradeoffs.

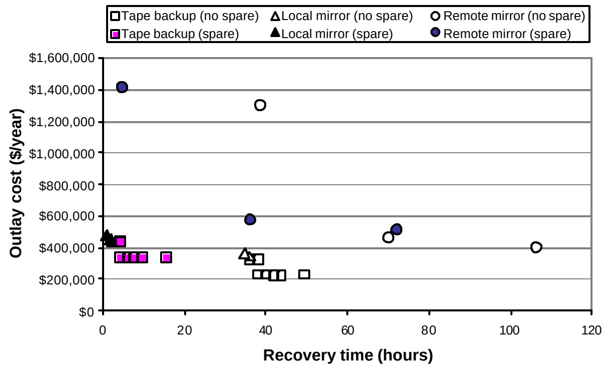

Figure 6: cost vs. recovery time tradeoffs.

For each design alternative, as we increase the resources used to support recoverability (e.g., network links or tape drives), cost increases –sometimes dramatically. As we add hot spare resources, outlay costs increase further. These outlay costs are dwarfed, however, by the penalty cost savings from the better recovery time that the additional resources provide. Tape backup penalty costs are dominated by data loss costs, so the addition of resources to improve recovery time behavior results in only minor penalty cost improvement.

Figures 5 and 6 provide a sensitivity analysis for the configuration parameters we examined. As expected, we observe that the addition of hot spare resources drastically reduces the recovery time for all of our design alternatives. Given that our time estimate for ordering, setting up and configuring a new array is optimistic, this disparity is likely to be even greater in practice. We observe that tape-based backup data loss varies greatly, depending on the length of the backup window and frequency of the backup operation. Data loss for the mirroring alternatives also varies considerably, depending on which variant is used: a synchronous protocol minimizes loss (0 GB), while an asynchronous logging protocol bounds data loss by the size of the cache set aside for logging (8 GB in our experiments), and an asynchronous batching protocol experiences a worst-case data loss that's twice the batching window. The huge range of recovery time behavior for remote mirroring is due to the provisioning of the wide area links, as described above.

Perhaps the most interesting result of this sensitivity analysis is that tape-based backup can be competitive with local mirroring for recovery time and data loss, given the appropriate configuration.

5. ANALYSIS

Nothing will ever be attempted, if all possible objections must be first overcome. – Samuel Johnson

Our experiences with this initial dependability design tool have helped us to understand several issues that weren't initially obvious to us.

Recovery-oriented design

It is all too easy to forget that the primary specification is recovery performance (e.g., the time to recover from a failure), not redundancy performance (e.g., the costs of maintaining redundant information). We "knew" this fact, but still found that this mistake crept into our thinking several times, such as in the determination of the number of wide area remote mirroring links and tape drives for backup, as described in the previous section.

Recovery metrics

Two metrics are commercially popular in this space: *recovery time objective*, *RTO* (how long does it take to get the data back?) and *recovery point objective*, *RPO* (how old will that data be?). The adoption of these metrics is clearly a step forward from more simplistic metrics like mean time to data loss (MTTDL), but they don't go far enough to describe many situations. For example:

1. It may be useful to "time travel" to different points in the past, rather than one, fixed interval. For example, it may be necessary to go back a month to find a secure copy of a vital file after a long-dormant security attack erupts; meanwhile, a user may want to go back to a version only a few minutes old to recover from an accidental delete error.

Recovery metrics should permit the specification of several recovery point goals: retrieving a particular data object from a past point (e.g., for deleted-file recovery), as well as restoring the entire system to some known prior point (e.g., after a site disaster).

2. Although we chose to ignore it in our current tool, there is a wealth of difference between "not down" and "operating adequately." Some applications can usefully operate at a fraction of their normal rate; others are essentially useless until close-to-normal performance is restored. Similarly, some applications can usefully operate with only a fraction of their data, whereas others require all data to be functional. This distinction affects the choice of both data redundancy techniques (e.g., it may mandate the use of snapshots and dedicated I/O ports to take backups), and recovery techniques (e.g., a RAID 5 degraded mode may never be acceptable, so the data will have to be fully replicated).

Recovery metrics should permit the administrator to describe whether these partially operational modes are desirable, or even possible.

Capturing costs

It is surprisingly hard to capture outlay and penalty costs in a meaningful way. This difficulty is partly the consequence of the rich design space, and (doubtless) partly because we are still exploring. Some of the questions we encountered are:

Outlay costs: What is the period over which a purchase is amortized? What about capital depreciation? service contracts? personnel overheads? installation costs? opportunity costs for idle equipment? How should shared equipment be charged (e.g., at an outsourced disaster-recovery site)?

Penalty costs: How should an outage be priced (e.g., the one-time worst-case cost, or that value multiplied by its frequency)? Who pays for replacement equipment? (And if it is the insurance company, is the cost of the equipment effectively zero?) How should data loss be priced? How do people price indirect costs, such as user dissatisfaction?

We expect that many of the outlay cost questions can be addressed by applying standard accounting principles, including amortizing costs over the useful lifetime of the system. We suspect that absolute penalty costs will be difficult to gather, at least at first, so we have geared our implementation towards a "try and see" interactive design tool that allows people to change the outage and data loss costs, and see the result. We also suspect that techniques common in the insurance industry will be helpful, such as "flash cards" that present the consequences of a failure.

Simultaneous exploration of objectives and solutions

We believe that, given a clear recovery time objective, it is relatively straightforward to triage the alternative mechanisms, eliminating those with too-high delays and (perhaps) unnecessarily good performance, and then explore the remaining subset of the design space. One problem this fails to recognize, though, is that most people are unable to provide a clear RTO, because they are making tradeoffs between the effectiveness of the solutions and their costs. Only when presented with both – and, perhaps, a sensitivity analysis of some of the alternatives – will they be able to come to a decision. The key observation is that these tradeoffs are hard to make – they involve too many alternatives and too many interactions for most people to consider, or even remember.

We believe that a tool that allows users to interactively explore the design space of possible objectives and solutions will be of great use – especially if it can take as input data in a form that is closer to the tradeoff space that people want to explore (e.g., recovery time, cost), rather than the list of techniques they feel the need to try.

Lack of orthogonality in the solution and failure spaces

As we hope was clear from the short taxonomy presented earlier, neither the failure nor the solution space are properly orthogonal. This causes interactions between solutions and failure types. For example, a disk array LU failure could result in failure of the primary data it stores, but could also disable the copy-on-write snapshots taken from it, and may (indirectly) influence the availability of spare resources.

One challenge in this space is to develop a concise way of capturing the effects of a failure, and the properties of a data protection technique. How exactly do you formalize what a remote mirror does? What about a restore from a tape backup? We suspect that, absent such formulations, failure- and solution-handling will remain a rather clumsy, handcrafted, skill-intensive activity.

Next steps

We found our initial forays into this space encouraging. As a result, we plan to continue this approach, extending our tool's functionality in a number of ways, including: more failure types and scopes (e.g., failure rates, concurrent failures; user, operator, and software faults), a greater range of dependability techniques, a choice of objective functions (e.g., how much dependability can be obtained for a fixed outlay), and a broader set of tradeoffs and business practices.

Although our initial prototype implementation has shown us many useful insights, the lack of automation in Excel prompts us to search for a new implementation vehicle for these next steps.

Finally, we suspect that this approach may help us identify "holes" in the solution space, which in turn may help guide the development of new data protection techniques.

6. RELATED WORK

The dependable systems community has techniques for modeling [8] and simulating [10] various aspects of dependability. The emphasis in this community appears to be on tools to assess designs that have been created by hand, as evidenced by their GUI interfaces (e.g., [6]). While this assessment is a necessary ability for any design system, it doesn't address the problem of how to explore the design space automatically.

System administrators have to make the design choices we describe here every day, but the focus in their literature is on operational issues, such as explaining backup policies and setting related configuration parameters.

The majority of "data protection" work is focused on developing (yet more) new protection mechanisms, with various twists and tradeoffs. Rarely do their designers consider the difficulties of deciding when to use them, and how they combine with existing techniques.

We *were* able to leverage existing work that deals with specifying and evaluating dependability requirements. Examples include a declarative method of specifying data performability and reliability requirements [11, 14, 15], and ways to measure the availability of RAID systems [5].

Some prior work from our own group has addressed the design of systems to meet performance goals (e.g., [4, 3]), but the design tools that resulted (e.g., [1, 3]) considered only online redundancy techniques (mirroring and RAID5). Our dependability design work builds on the experience gained there.

In short, we are not aware of any other work that shares our goal of automating data dependability design, nor of any system that accomplishes this goal.

7. CONCLUSIONS

The solution and failure spaces for dependable storage systems are rich, complex and inherently interesting. We believe that because of these issues, automating the design of dependable storage systems is likely to be an appropriate approach. Moreover, this approach is achievable – at least over the space we've tried so far. We described our simple dependability designer prototype, and presented initial results exploring the recovery time-data loss-cost space. The insights we came upon during this process suggest that there remains a wealth of interesting problems to explore in this space.

REFERENCES

The ability to quote is a serviceable substitute for wit. – Maugham

[1] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems* **19**(4):483–518, November 2001.

[2] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. Proceedings of *Conference on File and Storage Technologies (FAST)*, Monterey, CA, pages 175–188, January 2002.

[3] E. Anderson, R. Swaminathan, A. Veitch, G. Alvarez, and J. Wilkes. Selecting RAID levels for disk arrays. Proceedings of *Conference on File and Storage Technologies (FAST)*, Monterey, CA, pages 189–201, January 2002.

[4] E. Borowsky, R. Golding, A. Merchant, L. Schreier, E. Shriver, M. Spasojevic, and J. Wilkes. "Using attribute-managed storage to achieve QoS". Presented at *5th International Workshop on Quality of Service (IWQoS'97)*, Columbia University, New York, June 1997. Available from http://www.hpl.hp.com/SSP.

[5] A. Brown and D. Patterson. Towards availability benchmarks: a case study of software RAID systems. Proceedings of the *2000 USENIX Annual Technical Conference*, pp. 263–276, June 2000.

[6] D. D. Deavours, et al. The Möbius Framework and its implementation. *IEEE Transactions on Software Engineering*, **28**(10):956–969, October 2002.

[7] *Online survey results: 2001 cost of downtime*. Eagle Rock Alliance Ltd. Aug. 2001, accessed May 2003. Available from http://contingencyplanningresearch.com/2001%20Survey.pdf,.

[8] B. Haverkort, R. Marie, G. Rubino and K. Trivedi, eds. *Performability modeling: techniques and tools*, Wiley & Sons, May 2001.

[9] M. Ji, A. Veitch, and J. Wilkes. *Seneca: remote mirroring done write*. In Proceedings of the *USENIX technical conference* (San Antonio, TX), pp 253–268, June 2003.

[10] M. Kaaniche, et al. A hierarchical approach for dependability analysis of a commercial cache-based RAID storage architecture. Proceedings of the *28th International Symposium on Fault-Tolerant Computer Systems (FTCS-28)*, pp. 6–15, June 1998.

[11] K. Keeton and J. Wilkes. Automating data dependability. Proceedings of *10th ACM-SIGOPS European Workshop* (Saint-Emilion, France), pp. 93–100, Sept. 2002.

[12] Microsoft Excel Users' Guide, Version 5.0, Microsoft Corporation, 1994.

[13] The Data Recovery Solution. White paper by OnTrack Data Recovery, Inc., 1998, available from http://www.ontrack.com .

[14] J. Wilkes. Traveling to Rome: QoS specifications for automated storage system management. Proceedings of the *International Workshop on Quality of Service (IWQoS)*, pp. 75–91, June 2001.

[15] J. Wilkes and R. Stata. Specifying data availability in multi-device file systems. Proceedings of the *4th ACM-SIGOPS European Workshop*, September 1990; published as *Operating Systems Review* **25**(1):56-59, January 1991.

[16] R. Witty and D. Scott. Disaster recovery plans and systems are essential. *Gartner FirstTake FT-14-5021*, Gartner Research, Sept. 12, 2001.