Lessons and challenges in automating data dependability

Kimberly Keeton, Dirk Beyer, Jeffrey Chase, Arif Merchant, Cipriano Santos, and John Wilkes
Hewlett-Packard Laboratories, Palo Alto, CA
Duke University, Durham, NC
{kimberly.keeton,dirk.beyer,arif.merchant,cipriano.santos,john.wilkes}@hp.com, chase@cs.duke.edu

Abstract: Designing and managing dependable systems is a difficult endeavor. In this paper, we describe challenges in this vast problem space, including provisioning and allocating shared resources, adaptively managing system dependability, expressing dependability goals, interactively exploring the design space, and designing end-to-end service dependability. We outline the optimization-based approach we've used to tackle the data dependability portion of this space, and describe how we can extend that approach to address an even larger dependability scope.

1 Introduction

Dependability encompasses the sometimes competing goals of reliability, availability *and* performance. As a result, managing computer system dependability is even more difficult than managing performance. Techniques for improving availability and reliability often consume additional resources for redundancy, further contributing to the dilemma of how to allocate resources. Specifying dependability goals is challenging, and determining if they have been met is even harder. Failures are uncommon and unpredictable, and it is difficult to determine whether a particular system configuration can meet a probabilistically specified availability bound (e.g., five nines) for a given incidence of failures. Designing systems to meet these goals is also difficult: the design space of techniques for protecting information is surprisingly large, with myriad configuration choices. It's tough to understand how the techniques interact and how they impact recovery behavior after a failure.

Despite the vastness of this problem space, we have made progress towards the goal of automating the design of dependable data systems. At the 2002 SIGOPS European Workshop we explored ways to express dependability goals for automating storage system design [12]. Since then, we've realized that specifying dependability goals in financial terms (as penalties for downtime and data loss) allows us to evaluate cost vs. risk tradeoffs in a common currency. Our automated storage design tool uses this insight to formulate data dependability design as an optimization problem and employs quantitative models to explore the dependability implications of different storage design choices [11]. The tool determines what secondary copies should be made throughout the distributed storage hierarchy to protect the primary copy of a data set against various failures. It picks the appropriate data protection techniques (e.g., tape backup or remote mirroring) and recovery techniques (e.g., data reconstruction or site failover), as well as key configuration parameters, such as the frequency of update propagation, the hardware configuration, and the type of spare resources employed.

In this paper, we step back and outline challenges and opportunities in the larger space of managing dependable computer systems (Section 2). Section 3 describes our formulation for a small portion of the problem space, and how that solution can be adapted to address an even broader scope. Our goal is to encourage discussion about this exciting and important area, and to prompt an exchange of ideas about possible solutions. Section 4 concludes.

2 Defining the dependability landscape

Our conversations with consultants and customers indicate that information system designers and administrators have a wide range of needs when managing system dependability. Broadly speaking, these needs include coping with complex, dynamic systems; expressing goals and understanding choices; and holistically approaching dependability. In the following sections, we describe these needs in more detail.

2.1 Coping with complex, dynamic systems

Dealing with complex systems that may change over time requires the ability to provision, allocate and share resources, adaptively manage dependability, and extensibly support data protection techniques, as shown in Figure 1.

Provisioning, allocating and sharing resources. Shared resource environments such as utility data cen-

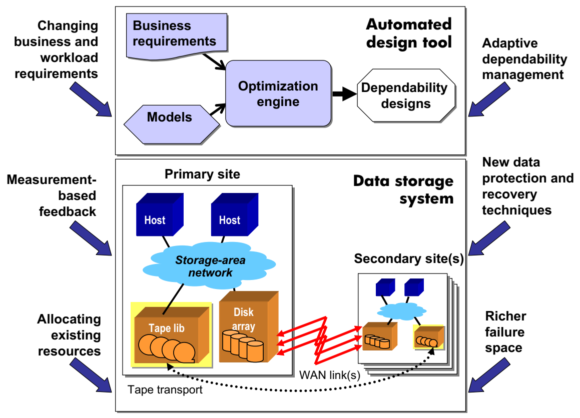

Figure 1: Extensions for managing dependability in complex, dynamic systems.

ters [6, 7, 9, 14, 16] and disaster recovery sites are becoming more popular. They require the ability to provision resources and allocate them among competing customers. Customers who manage their own computing and storage infrastructure must also balance allocations between competing workloads. For instance, some multi-national companies employ multiple primary sites on different continents, which serve as secondary sites for one another; each site must allocate its resources between the local primary workload and the remote secondary workload.

Both customers and service providers want to design systems that will protect against a range of failures, not just cope with single failure scenarios. In addition to "green field" provisioning, they also want to find the best way to allocate their existing resources.

Extensible support for data protection techniques. Technology continues to offer us an ever-growing set of choices for protecting data. Traditionally, data dependability for the primary copy was maintained through RAID techniques to withstand disk failures, and secondary copies (backups) made on magnetic tapes. We now also have the ability to create local online copies of the data, remote replicas maintained through synchronous or asynchronous updates, and backups onto disk, optical disk and a variety of tape technologies. Moreover, these methods can be combined to meet requirements that a single technique cannot achieve. For instance, to meet regulatory requirements while avoiding degraded performance, system designers may employ synchronous mirroring to a local site in concert with asynchronous mirroring between the second (local) site and a third remote site. Alternatively, data may be mirrored to a secondary site, where it is backed up to tape. In all cases, system designers must be able to understand the alternatives' recovery behavior sufficiently to pick the most appropriate solution among the different techniques, including choosing the appropriate configuration parameters (e.g., number of wide area links, asynchronous mirroring write absorption interval). They also want to take advantage of new technologies as they become available.

Adaptive dependability management. Dependable system design and initial deployment are the first steps in managing the system as it evolves in response to changing business and workload demands, component upgrades and failures. Once a storage system has been deployed, system administrators and users want assurance that the system design will respond to failures as expected (e.g., with reasonable recovery time and data loss). Should business or workload demands evolve, they want to reallocate and/or reprovision resources to accommodate these changes. Should a failure occur, they need to determine whether and how the workloads can continue running on the resources that remain. An important consideration is the cost of recovering workloads from the failed resources onto the available resources. Because the available resources may not be sufficient to support the original workloads, system administrators must be able to decide which workloads to stop to "make room" for the most critical workloads. (These stopped workloads can later be restarted if additional resources can be provisioned.)

2.2 Expressing goals and understanding choices

Provisioning, allocating and adaptively managing resources are critical for ensuring the dependable operation and evolution of the underlying system. Interestingly, interacting with customers to gather requirement information and to show them the financial implications of their choices may be nearly as important. Figure 2 illustrates this iterative process.

Collecting meaningful goals. Data dependability design is predicated on the ability to collect meaningful customer goals. The better (e.g., more quantitatively) customers can specify their goals, the better the chance that the resulting system will meet those goals. Unfortunately, customers frequently can't state their goals quantitatively. They have an even harder time describing quantitative utility functions that ascribe financial implications to their desired service levels.

Consultants currently use various techniques, including business continuity assessments [5], to determine a customer's readiness to handle a disaster and to collect qualitative information on current management and disaster recovery practices. These tools are used to estimate business continuity goals such as recovery time and recovery point objectives (RTOs and RPOs) [4]. The RTO

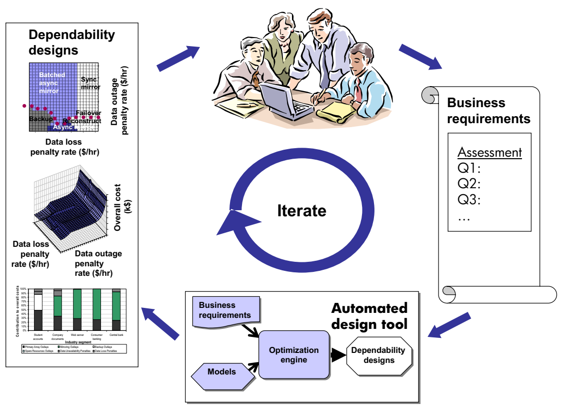

Figure 2: Iterative process for expressing goals and understanding choices.

provides an upper bound on the elapsed time after a failure before a business service (e.g., application) is up and running again. When a failure occurs, it may be necessary to revert back to a consistent point prior to the failure, which will entail the loss of any updates after that point. The RPO provides an upper bound on the recent updates (expressed in time) lost during recovery from a failure. Although these metrics do provide quantitative recovery goals, they provide no insight into the financial ramifications of coping with failures. Designing the right tools (e.g., questionnaires, graphical user interfaces) to query customers for quantitative, financial business goals remains an open problem.

Richer goal specifications. Penalty rates [11] provide an alternative abstraction for representing customers' business requirements for downtime (US dollars per hour downtime) and data loss (US dollars per hour of lost recent updates). However, linear penalty rates may be insufficient to describe customers' requirements. For instance, the first five minutes of downtime may be inexpensive, the next hour slightly more expensive, the next 8 hours very expensive, and so on — all the way to bankruptcy in the limit. Downtime at different times of the day (or week or month) may also be penalized differently. Additionally, failures resulting in recent data loss at different points in the secondary copy creation cycle may be valued differently. For example, a failure during the last minute before a central bank synchronizes with the US Securities and Exchange Commission is more critical than a failure in the first minute after the synchronization has completed. What's more, system designers may want the ability to express future growth trends, as well as more intangible goals, such as minimizing staff training requirements, design simplicity, or design conservatism to handle risk aversion (e.g., "the first failure means no bonus; the second failure means no job").

Exploring the design space. Customers may not understand the ramifications of the design choices they make. In some cases, they may not be able to quantify even simple dependability goals. A toolset that can easily navigate the design space to evaluate "what if" scenarios would be quite useful to increase customer understanding. Even if they can't quantify their requirements, the ability to interactively explore the design space may show the boundaries between different operational regions, where different solutions prevail. Customers may find choosing between regions (e.g., ranges of input requirements) easier than picking a particular point in the requirements space.

2.3 Holistic approaches to dependability

Making data storage systems dependable is a critical first step towards the goal of provisioning overall system resources to meet customers' quality of service goals. We consider separately the questions of end-to-end dependability design and multi-attribute provisioning.

End-to-end dependability design. Upon failures, the entire infrastructure must recover to permit the service to continue operation. As a result, ultimately users want to provision server, networking, middleware and application resources to meet their end-to-end service dependability goals. This more holistic approach means that the space of potential dependability techniques is expanded to include mechanisms at other levels of the application stack, including snapshots, checkpointing, logging, replication and failover at the file system, database and application levels, as well as interactions between these mechanisms. For example, Total Recall dynamically sets the number of file replicas according to a per-file availability target for given site failure rates in a peer-to-peer system [3]. The batch-aware file system dynamically determines whether to checkpoint application state to a temporary file to avoid restarting computations if a task fails [2]. The AVED project examines how to automatically provision computational resources between tiers in a multi-tier environment to support availability and performance goals [8].

Multi-attribute provisioning. Although most research in automating design decisions has focused on performance alone [1] or on availability and reliability [8, 11], customers ultimately want to design systems that will meet all of their goals, including reliability, availability, performance and security. The notion of performability — performance at a given level of availability — partially allows customers to describe their needs. For example, the system should perform at "at least 200 MB/sec 25% of the time, at least 150 MB/sec 50% of the time, at least 100 MB/sec 95% of the time and 0 MB/s no more than 5% of the time."

Given that the price tag for a system that meets all of the customer's goals may be too high, customers also need a way to express tradeoffs between the attributes, and to consider these tradeoffs when designing the system. For instance, a customer may want to achieve a base level of reliability before worrying about performance and availability, and then a base level of performability before balancing these concerns (e.g., "achieve reliability to level A, followed by performability to level B, followed by an even mix up to levels C and D.") More generalized utility functions may provide a method for expressing such tradeoffs [13].

3 Automating data dependability

Although the problem space described in the previous section is very rich, we have made considerable progress in tackling a limited portion of that space. In [11], we describe the architecture and evaluation of an automated tool for designing dependable storage systems. It combines quantitative models of the most common data protection and recovery techniques — remote mirroring, tape backup, data reconstruction and failover — with an off-the-shelf optimizing solver to choose the best alternative.

Customers provide business goals expressed as financial *penalty rates*: US dollars per hour of outage time (unavailability) and US dollars per hour of data loss, specified as how much recent data (in time) is lost during the recovery process. By expressing business requirements as financial penalties, we can cast the overall design question as an optimization problem, using mathematical programming techniques (e.g., mixed-integer programming [17]). The optimization objective is to minimize the *overall business costs*, defined as *outlays* plus *penalties* for a failure of the primary copy of data (e.g., due to array failure or site disaster). Outlays include equipment and facilities costs for the primary array(s), data protection techniques, failover alternatives and spare resources. Penalties are associated with the outage time and data loss for recently-written data. The outputs of the design tool are the storage solution design, the estimated worst-case data loss and recovery times, and the outlay and penalty costs. In addition to selecting the data protection and recovery techniques with the least overall cost, the solution includes settings for parameter values, such as the number of tape drives or network links, the duration of backup windows, and the type of spare resources employed (e.g., hot vs. cold, dedicated vs. shared).

The resulting optimization tool can be used for a number of different purposes, including evaluating the dependability and cost of existing designs, determining the best solution for the specified business inputs, and exploring the design space across a wide range of business requirements.

Narrowing the scope of the problem allowed us to make considerable progress, at the cost of somewhat restricted functionality. For simplicity, we treat all workload data objects as having the same goals. The current optimization framework focuses on the question of how to provision resources for "green field" designs, without considering how to allocate or augment an existing pool of resources. It performs a worst-case scenario analysis by considering the effects of a single failure at a time, resulting in designs that focus on that failure type, to the exclusion of others. For ease of prototyping, we implement the models of data protection technique behavior directly in the optimization framework. However, this monolithic structure makes it difficult to add new data protection techniques, or to combine existing ones.

Our experience leads us to believe that the general approach of treating dependability design as an optimization problem holds promise for addressing an even larger portion of the dependability management space. In this section, we outline how our existing framework can be extended in a straightforward fashion to answer some of the challenges raised in Section 2.

3.1 Allocating shared resources

Addressing challenges in allocating and sharing already-provisioned resources requires several extensions to our automated design infrastructure:

Multiple workload objects. Provisioning and allocating resources in shared environments requires support for multiple workload objects, including the ability to specify independent requirements for different workload objects and the ability to describe dependencies between workload objects. (For example, the ability to access user data depends on the availability of operating system and file system data.) The models and optimization framework must now consider not only what resources are provisioned, but also how the workload objects are allocated to resources and how the objects depend on one another. The framework must also be able to evaluate penalty functions separately for each workload object.

Allocating existing resources. The optimization support for allocating an existing set of resources is straightforward: we merely fix the values for decision variables corresponding to those resources (e.g., number of tape drives, wide area mirroring links, or disk arrays). The same objective function used in the "green field" design case can then be evaluated to determine the allocation that minimizes the overall costs.

Expected system dependability. Computing expected dependability under a broad range of different failures requires support from both the models and the solver. First, the models must be able to estimate the recovery time and recent data loss under each failure type of interest. Examples include primary array/site failures, partial primary failures, data protection technique failures (e.g., mirroring link(s), tape drive(s) or tape media), and multiple simultaneous failures (e.g., a failure scope large enough to take out both primary and secondary copies).

The solver needs the ability to consider multiple potential scenarios and weight them appropriately. Fortunately, robust optimization techniques [15] can be applied. Briefly, a separate optimization model is constructed for each possible failure scenario, with localized decision variables that correspond to decisions that must be made only under that scenario. Decisions that are independent of the failure scenario are represented by decision variables common to all of the models. The overall optimization model incorporates all of the scenario-specific constraints, and computes the overall objective function by appropriately weighting the values of the individual objective functions. This approach has the added benefit that it's easy to incorporate a new failure scenario into the solver.

Choosing the appropriate weighting factors is challenging: merely using failure occurrence probabilities will discount the potentially devastating losses that occur for very infrequent, but large-scale failures. We believe that it may be more appropriate to use relative failure probabilities or perceived failure importance. The latter option permits users to express their risk aversion, and as a result, more fully consider the magnitude of losses accompanying disasters.

3.2 Extensible modeling support

We deal with the ever-increasing list of data protection techniques by providing a framework in which models of individual dependability techniques can be composed [10]. The framework uses a common set of parameters to describe the most popular data protection techniques, which enables the addition of models for new techniques as they are invented. Because the detailed models of each technique's operation are maintained separately from the composition framework, it is also straightforward to incorporate more sophisticated models of existing techniques.

An open question in this space is how to describe reasonable combinations of data protection techniques. For instance, to create a consistent tape backup, one needs access to a point in time copy such as a snapshot or array-based split mirror. There may also be constraints on which order techniques may be applied: for example, it makes sense to mirror data from the local array to a remote array, and then create a tape backup at the remote site, but not the other way around. Such a "grammar" would describe valid inputs to the modeling framework, and also inform the solver's need to express feasible solutions.

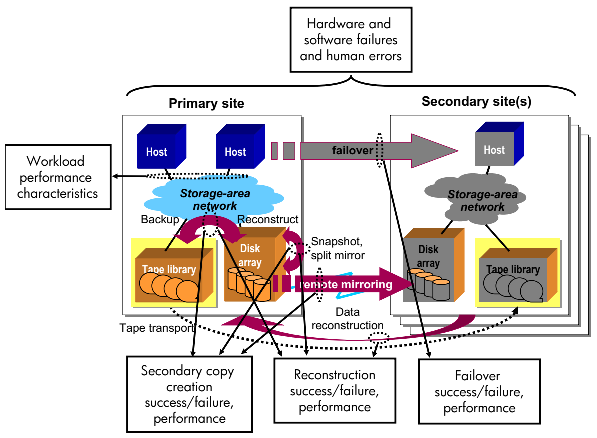

Figure 3: Instrumentation to support adaptive dependability management.

3.3 Richer goal specifications

To support more complex business requirements, we must determine how to quantitatively specify them and how to extend the models and solver to incorporate the more detailed requirements. For instance, we could use piecewise linear penalty rates to capture penalties that increase as a function of the length of outage or recent data loss duration. This approach permits us to assign very large penalty rates to impose hard constraints on recovery time and data loss (e.g., corresponding to the RTO and RPO values used by consultants today). In this case, the solver would be extended to assess penalties as a function of outage or data loss duration. Capturing qualitative or intangible requirements, such as design simplicity, remains an open problem.

3.4 Adaptive dependability management

Analogous to the performance provisioning case [1], we envision that our solver could be included in an adaptive management cycle to (re)design, (re)configure and analyze the dependable system throughout its lifecycle.

To support reprovisioning or augmenting system resources, we can extend the solver. To expand the existing resources to support increased demand, we can use the same objective function as for "green field" design, but dramatically lower the price of existing resources, to ensure they are used to their fullest capacity. Reallocating resources in response to failures or disasters is somewhat more complicated. In addition to evaluating the best allocation for the reduced set of resources, the solver must also capture the costs of recovering the workloads from the failed resources onto the available resources.

Once the storage system has been deployed, the management system should be able to verify that the design's achieved behavior meets the behavior predicted by the modeling framework. As shown in Figure 3, the management system can measure the normal mode behavior of data protection techniques to quantify the performance of successful secondary copy creation (e.g., incremental tape backup). It can measure recovery behavior after actual failures (e.g., accidental deletions by users, array component failures). It can also measure recovery behavior under disaster recovery drills to verify correct operation under more severe and infrequent failures. If sufficient performance and capacity exist in the system, the management system could also proactively inject smaller-scope faults to test recovery behavior.

These measurements of normal behavior and recovery time and data loss under failures can provide feedback to the modeling framework, to help adjust its estimates for future designs. The management system can additionally collect information on the frequencies of various types of failures, to inform its failure models. Finally, to fine-tune the solver's workload models, the management system can also measure the update characteristics of the workloads running on the system.

4 Conclusions

Although automating the design of dependable storage systems might have seemed an infeasible goal upon first consideration, we've demonstrated that it is an achievable one. In this paper, we've articulated several research opportunities for managing dependable systems. We've also described one potential approach — optimization based on financial objectives — for addressing these challenges. By outlining this vision, we hope to entice other researchers to join us in exploring this exciting space.

References

[1] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. In *Proc. 1st Conf. File and Storage Technologies (FAST)*, pages 175–188, Monterrey, CA, USA, January 2002.

[2] J. Bent, D. Thain, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Explicit control in the batch-aware distributed file system. In *Proc. 1st Symp. on Networked Systems Design and Implementation (NSDI)*, pages 365–378, San Francisco, CA, USA, March 2004.

[3] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. Total Recall: System support for automated availability management. In *Proc. NSDI*, pages 337–350, San Francisco, CA, USA, March 2004.

[4] C. Brooks, M. Bedernjak, I. Juran, and J. Merryman. *Disaster recovery strategies with Tivoli storage management*. IBM International Technical Support Organization, version 2 edition, November 2002.

[5] Hewlett-Packard Company. Free business continuity readiness assessment. http://www.hp.com/large/promo/bcsurvey/index.html, Last accessed September 2004.

[6] Hewlett-Packard. Utility data center. http://www.hp.com/large/infrastructure/utilitydata/overview/, Last accessed September 2004.

[7] IBM. On demand business. http://www.ibm.com/ondemand, Last accessed September 2004.

[8] J. Janakiraman, J. R. Santos, and Y. Turner. Automated system design for availability. In *Proc. Intl. Conf. on Dependable Systems and Networks (DSN)*, Firenze, Italy, June 2004.

[9] D. Kandlur and J. Killela eds. Utility computing. *IBM Systems Journal special issue*, 43(1), 2004.

[10] K. Keeton and A. Merchant. A framework for evaluating storage system dependability. In *Proc. DSN*, pages 877–886, Firenze, Italy, June 2004.

[11] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. Designing for disasters. In *Proc. 3rd FAST*, pages 59–72, San Francisco, CA, USA, March 2004.

[12] K. Keeton and J. Wilkes. Automating data dependability. In *Proc. 10th ACM-SIGOPS European Workshop*, pages 93–100, Saint-Emilion, France, September 2002.

[13] C. Lee. *On quality of service management*. Technical report CMU-CS-99-165, Carnegie Mellon University, August 1999.

[14] Sun Microsystems. N1 grid — introducing just in time computing. http://wwws.sun.com/software/solutions/n1/wp-n1.pdf.

[15] H. Vladimirou and S. Zenios. *"Stochastic Programming and Robust Optimization" in Advances in Sensitivity Analysis and Parametric Programming, edited by T. Gal and H. Greenberg*, chapter 12. Kluwer Academic Press, 1997.

[16] J. Wilkes, J. Mogul, and J. Suermondt. Utilification. In *Proc. of the 11th ACM SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[17] L. Wolsey. *Integer Programming*. John Wiley and Sons, Inc., 1998.