

On the Road to Recovery: Restoring Data after Disasters

Kimberly Keeton, Dirk Beyer, Ernesto Brau, Arif Merchant,
Cipriano Santos and Alex Zhang

Hewlett-Packard Labs, 1501 Page Mill Rd. M/S 1134, Palo Alto, CA 94304, USA

+1 (650) 857-3990 (voice); +1 (650) 857-5548 (fax)

firstname.lastname@hp.com

Abstract—Restoring data operations after a disaster is a daunting task: how should recovery be performed to minimize data loss and application downtime? Administrators are under considerable pressure to recover quickly, so they lack time to make good scheduling decisions. They schedule recovery based on rules of thumb, or on pre-determined orders that might not be best for the failure occurrence. With multiple workloads and recovery techniques, the number of possibilities is large, so the decision process is not trivial.

This paper makes several contributions to the area of data recovery scheduling. First, we formalize the description of potential recovery processes by defining recovery graphs. Recovery graphs explicitly capture alternative approaches for recovering workloads, including their recovery tasks, operational states, timing information and precedence relationships. Second, we formulate the data recovery scheduling problem as an optimization problem, where the goal is to find the schedule that minimizes the financial penalties due to downtime, data loss and vulnerability to subsequent failures. Third, we present several methods for finding optimal or near-optimal solutions, including priority-based, randomized and genetic algorithm-guided ad hoc heuristics. We quantitatively evaluate these methods using realistic storage system designs and workloads, and compare the quality of the algorithms' solutions to optimal solutions provided by a math programming formulation and to the solutions from a simple heuristic that emulates the choices made by human administrators. We find that our heuristics' solutions improve on the administrator heuristic's solutions, often approaching or achieving optimality.

Categories and Subject Descriptors: D.4.5 Operating systems: Reliability, K.6 Management of computing and information systems, G.1.6 Numerical analysis: Optimization

General Terms: Management, reliability, algorithms, design

Keywords: Data storage, disaster recovery, backup/restore, management, scheduling, optimization, math programming, genetic algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EuroSys '06, April 18–21, 2006, Leuven, Belgium.
Copyright 2006 ACM 1-59593-322-0/06/0004 ...\$5.00.

1. INTRODUCTION

“In the midst of a disaster, the recovery team is under stress. That’s not the time to be exercising seldom-practiced procedures that could be made automatic.” —Glen Bellomy, Veritas System Engineer [25]

Disasters aren't limited to naturally occurring events like earthquakes, fires or floods. Any event that leads to a loss of data or the inability to access it can have disastrous consequences, such as lost worker productivity, lost revenue, damaged corporate reputation or even bankruptcy. Under such circumstances, system administrators must work quickly to restore operations, hopefully minimizing data loss and vulnerability to subsequent failures during recovery. The cost of inefficiency is high: one hour of unnecessary downtime can cost enterprises hundreds of thousands or millions of dollars.

Today administrators often use rules of thumb, serializing recovery of workloads based on the workloads' criticality to the business. To minimize the chaos that ensues after an event, many businesses institute disaster recovery plans or business continuity plans, which describe what steps to take after a disaster, as well as who is responsible for completing these tasks [25]. Recovery plans centralize information from a variety of disparate sources about the IT system's structure and how to recover it. These plans are designed ahead of time, and

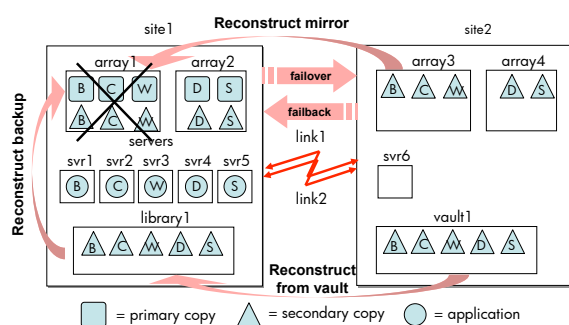


Figure 1: Data recovery scheduling problem. Application data may be protected by a combination of techniques, such as snapshots, remote mirroring, tape backup and remote vaulting of tapes. Applications have associated penalty rates for outage duration, recent data loss and vulnerability to additional failures. In the event of a failure that affects multiple applications, the application recovery operations share resources. We must choose which copy to recover for each application, and determine a schedule for the recovery operations that minimizes the overall penalty costs.

thoroughly tested, to ensure that personnel are familiar enough with the procedures to be able to execute them smoothly during a highly stressful time. This process is a lengthy one, meaning that updating the plan to handle changes in system configuration, business requirements, or service providers is non-trivial. Teams generally develop a handful of specific plans, to respond to more likely disaster scenarios (e.g., a hurricane in the Gulf of Mexico), and a more general plan to handle all other disasters. As a result, the plan that is executed to recover from an unanticipated disaster may not be tailored to best handle the failure scenario.

The focus of this paper is the *data recovery scheduling problem*, as illustrated in Figure 1. This problem includes determining what copy or copies of data should be used to recover each workload, how recovery operations should be balanced with each other and with the continued execution of workloads unaffected by the failure, and how device resources should be scheduled. With multiple workloads to recover and multiple strategies for accomplishing each workload's recovery, the number of possible recovery schedules is large. Because of these factors, administrators find it difficult to optimize their scheduling of recovery tasks to best respond to the event at hand.

This paper makes several contributions. First, we formalize the description of potential recovery processes by defining *recovery graphs*. Recovery graphs explicitly capture alternative approaches for recovering workloads, including their recovery tasks, operational states, resource requirements and precedence relationships. Solution architects who design disaster recovery plans tell us that recovery graphs provide more structure for gathering relevant information and defining recovery plans. Recovery graphs also provide a framework for quantitatively evaluating candidate recovery plans to determine their potential effectiveness and permit comparisons. They may also help in rehearsing plans, either piecemeal or in their entirety, to debug the plans themselves (e.g., have recovery operations been missed?) and to increase administrator familiarity with recovery steps. Finally, they may be helpful in evaluating the success rate and execution time for different recovery operations during rehearsals.

Second, using this formal description, we formulate the data recovery scheduling problem as an *optimization problem*, where the goal is to find the recovery schedule that minimizes the financial penalties due to application downtime, loss of recent data, and vulnerability to subsequent failures. An optimization-based approach produces higher-quality recovery schedules than administrators' rules of thumb, because it can consider a larger number of possibilities. Recovery schedules could even be generated on the fly, so that they could be tailored to the actual scenario that occurred. An automated, optimization-based approach produces the best recovery schedule quickly, simplifying the initial recovery plan design and plan maintenance as the environment evolves. The recovery solver may even be used to compare the recovery behavior of different storage solutions, to inform the design of a dependable storage system [21].

Third, we present several methods for addressing the data recovery scheduling problem, including priority-based, randomized and genetic algorithm-guided heuristics. We compare the quality of the solutions produced by these methods to the optimal solutions provided by a math programming for-

mulation, and to the solutions from a simple heuristic that emulates administrator behavior. We find that the administrators' categorical approach does not provide optimal solutions, with the inefficiencies resulting in millions of dollars of extra penalties in the face of disasters. Although simple, greedy approaches may be effective if resource constraints are loose, these approaches do not succeed in more tightly constrained environments. Randomization helps, if the algorithm is permitted to search a sufficiently large portion of the solution space. The best overall performance for our case studies is provided by an algorithm that uses a genetic algorithm to choose the recovery alternative and a greedy approach for determining the schedule. This algorithm produces results sufficiently quickly that it could be used to provide recovery schedules on the fly.

The rest of this paper is organized as follows. Section 2 more formally defines the data recovery scheduling problem, and introduces recovery graphs. Section 3 presents our formulations of the problem, and describes our prototype implementations. Section 4 provides quantitative experimental results for several example storage system designs and recovery scenarios. Section 5 summarizes related literature and Section 6 concludes.

2. DATA RECOVERY SCHEDULING PROBLEM

In this section, we more formally define the data recovery scheduling problem, and introduce recovery graphs as an abstraction for representing the problem.

2.1 Background

We assume an environment with multiple sites and multiple applications, where each application's persistent data is protected by one or more data protection techniques, as illustrated in Figure 1. Standard solutions for protecting data include intra-array redundancy (e.g., RAID techniques [27] and point-in-time copies [2] like split mirroring and space-efficient snapshots), inter-array mirroring (local and remote, synchronous and asynchronous) [17, 30], backup (e.g., to traditional tape systems or faster disk-based systems) [7, 14, 37], and remote vaulting. Replication and erasure coding are used in distributed storage systems such as PASIS [34] and FAB [29] to maintain data availability despite component failures.

In all cases, a primary copy of the data is protected by making one or more secondary copies, which are isolated from failures that may affect the primary copy. Several techniques may be combined to protect a given application. For instance, a combination of snapshots, tape backup and tape vaulting may be used to protect against user error and provide archival storage, and synchronous mirroring may be used to protect against site disaster.

In the event of a disk array failure or site disaster, multiple applications may be within the scope of the failure and subsequently require recovery. Recovery involves either data restoration from a secondary copy at the primary site or a secondary site, or site failover to a secondary mirror. For the former, data is copied from the secondary copy to the target site; for the latter, the computation is simply transferred to the secondary mirror, without any data copy operations. Failover requires a later failback operation (performed in the background)

to copy data to the target site. The recovery process is completed once the system has returned to its normal (i.e., pre-failure) mode of operation — all application workloads are running, and all of their data protection workloads have been resumed.

Failed applications incur penalty costs due to recovery time and lost recent data (because the secondary copy chosen as the recovery source may be out-of-date). Additionally, applications otherwise unaffected by the failure may be stopped (or their performance degraded) temporarily to free resources for recovering failed applications, thus incurring outage penalties. The *data outage* and *recent data loss penalty rate* input requirements describe the rate (in US\$ or Euros/hour) at which penalties are incurred for data unavailability and recent data loss, respectively. The data protection workloads for the unaffected applications may also be stopped (or degraded) to free resources for recovery. This degradation may result in an increased vulnerability to further, future failures affecting this application. We capture this increased potential for loss with the *data vulnerability penalty*, which is the worst case recent data loss penalty the application might incur for subsequent failures, discounted by the (conditional) likelihood of those failures.

The goal of data recovery scheduling is to find a schedule that minimizes the overall penalties — data outage, loss and vulnerability penalties — across all applications. The key questions to be answered include:

Which secondary copy should be used to recover each failed application? Tradeoffs exist between different secondary copies: for instance, the one that provides the smallest recent data loss may take longer to recover than an alternate copy that provides greater data loss and faster recovery time (e.g., depending on resource availability due to competition with other recovery or unaffected workloads). The correct choice depends on the relative importance of recovery (i.e., outage) time and recent data loss, as given by the penalty rates.

How should otherwise unaffected applications be handled? Recovery workloads will share resources with workloads unaffected by the failure — both the application workloads, which read and write the primary copy of the data, as well as the data protection workloads used to make secondary copies for these workloads. If the outage penalty rates for the unaffected workloads are low relative to those of the failed workloads, we may choose to degrade or shut down the unaffected application workloads and/or their data protection workloads, in favor of speeding the recovery of failed workloads.

We note that degrading an application's performance requirements will also reduce its protection technique requirements (e.g., a reduced update rate means fewer updates to copy). Degrading or stopping the data protection workloads, while the application runs at normal performance, may result in increased vulnerability to future failures, however.

How should device resources (e.g., disk arrays, tape libraries, wide area network links) be scheduled to support recovery operations and unaffected applications? Answers to the first two questions determine the set of operations to be performed to complete recovery. We must determine how to schedule these operations to avoid oversubscription of the underlying resources. Many of these resources

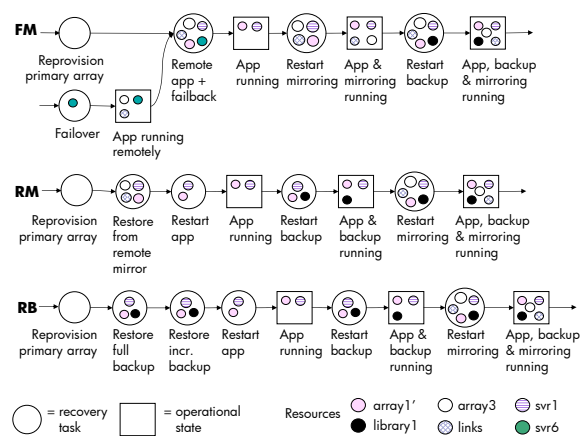


Figure 2: Partial recovery graph for a workload impacted by a primary array failure. This figure describes three recovery alternatives and their recovery operations for the “B” (commercial banking) workload shown in Figure 1, in the event of a primary array (array1) failure. The recovery paths correspond to the following recovery strategies: failover to remote mirror (FM), restore from remote mirror (RM), and restore from backup (RB). A full recovery graph includes recovery paths for all workloads in the system.

can easily be shared between multiple workloads. This approach would allow recovery to proceed in parallel with the continued operation of unaffected applications, or allow multiple recovery workloads to share a resource concurrently. Alternately, we may choose to allocate resources to a single workload at a time to permit individual recovery actions to complete sooner; in this case, we must decide the order in which the different workloads should hold a resource.

2.2 Recovery graphs

To facilitate the development of automated scheduling algorithms, we want to formalize the representation of the data recovery scheduling problem. A *recovery graph* captures different alternatives for recovering application datasets for a particular failure scenario. Figure 2 presents a partial recovery graph for one of the applications from Figure 1 after a primary array failure. Each recovery strategy (e.g., recovery from a secondary copy with a particular set of resources) is encoded as a path in the graph. The graph may contain multiple paths per application, depending on how the application's data was protected. For instance, the paths in Figure 2 correspond to failing over to a remote mirror (FM), restoring the dataset from a remote mirror (RM), and restoring from a backup copy (RB).

Each recovery path consists of a set of operations (also called “jobs”) to recover the application to its normal mode of operation, including restoring the data, restarting the application and restarting the data protection workloads. Some jobs must be completed before others may start (e.g., data can only be reloaded onto a failed array after the array has been repaired or replaced). The recovery graph captures these operations as a set of nodes, connected by directed edges that express precedence relationships. For instance, the restore from backup path includes the following steps: reprovisioning the primary array; restoring data from the full

backup; restoring data from the incremental backup; restarting the application; and resuming backups and mirroring to bring the workload into its pre-failure state of protection.

Recovery jobs demand resources. For storage devices, we assume that resources correspond to device instances (e.g., recovery of a mirror copy can only proceed from the specific array holding that copy). For network and computational devices, a resource may be a specific instance or a “virtual” device corresponding to the aggregate capabilities of an underlying class of devices (e.g., a network with an aggregate bandwidth of 40 MB/s). We assume that resources may be shared between different workloads. Resources have capabilities (e.g., bandwidth, capacity), which must be explicitly specified, to permit enforcement of resource constraints. The sum of all demands for a given resource cannot exceed its capabilities.

Recovery graphs distinguish between two types of jobs: *recovery tasks* and *operational states*. Recovery tasks correspond to a fixed amount of work (e.g., restoring a full backup copy or reprovisioning a resource). Their execution time depends on the allocation of resources required to accomplish the work. For instance, restoring a one TB full backup using 60 MB/s of tape drive bandwidth takes nearly five hours; it would take nearly ten hours at 30 MB/s. For tasks that demand no IT resources (e.g., overnight shipment of tapes from a remote vault), the execution time is modeled as a constant delay. After the work has been completed, recovery can transition from this node to the next node along the path.

Once application data has been restored, the application and its data protection workloads are eligible for resumption. Recovery graph nodes describe the operational states that recovery must pass through to restart all of these workloads. Each state’s resource demands are the minimum resources necessary to support the workloads that are active during that state (e.g., the application’s access rates if the application has been restarted).

In Figure 2, recovery tasks are represented by circles and operational states by squares. The smaller, shaded circles inside the nodes correspond to resources demanded by that job. We assume that each job’s resource demand is quantified as an annotation to the node in the graph.

2.3 Problem statement

Given the definition of recovery graphs, we can more formally define the data recovery scheduling problem. The inputs to the problem include the following: a set of outage, loss and vulnerability penalty rates for each application workload; a set of capabilities (e.g., bandwidth) per resource; and a recovery graph describing alternate recovery paths for each workload, including their jobs, the jobs’ resource requirements and their precedence relationships.

As stated above, our goal is to find a solution that minimizes the total penalty costs incurred by all workloads. The solution includes a) the choice of a recovery strategy (e.g., recovery path) for each workload and b) a schedule of jobs. A feasible schedule is constructed by picking a recovery path for each workload, and then scheduling the jobs in the selected paths so that both precedence constraints and resource constraints are met.

3. SOLUTION TECHNIQUES

In this section, we describe several algorithms for solving the recovery scheduling problem. We begin by establishing baselines for subsequent evaluation: Section 3.1 presents a heuristic to approximate the ad hoc approach used by human administrators, and Section 3.2 describes a mixed integer programming (MIP) formulation, which provides optimal solutions for small problem sizes. We then present several heuristic approaches in Section 3.3.

3.1 Human administrator heuristic

Human administrators approach recovery scheduling by classifying workloads into a small number of categories (e.g., platinum, gold, silver, bronze), based on coarse-grained estimates of sensitivity to data outage, loss and vulnerability. Workloads are recovered by category, starting with the highest priority category: all failed workloads in one category are recovered (and unaffected workloads restarted) before beginning the next category.

The Administrator heuristic captures this approach using five categories, one for each order of magnitude of the applications’ data outage penalty rate input requirements, ranging from \$100/hour to \$1M/hour. For each workload, the heuristic picks the recovery path that leads to the least recent data loss (e.g., remote mirroring is chosen over tape backup/vaulting). If multiple such paths exist, the algorithm favors the one that leads to the shortest recovery time for an environment with infinite resources (e.g., failover is preferred over reconstructing the mirror).

Within a category, multiple workloads may be simultaneously recovered, sharing access to resources. Within each category the heuristic builds the schedule by choosing jobs in descending penalty rate order from the list of eligible jobs, and continuing to schedule jobs as long as sufficient resources exist. Ties are broken by favoring jobs from recovery paths for failed workloads over jobs from unaffected workloads. After applying these rules, any remaining ties are broken by choosing a random job.

Optimizations to this algorithm are possible. For instance, if a recovery path chosen for a workload in a lower priority class uses a disjoint set of resources from the chosen paths in higher priority categories, it may proceed. Sharing resources between workload recovery in different classes is not permitted, however. The lower priority class path must wait until the higher priority class path has completed its use of the shared resources for recovery before the lower priority class path can start.

3.2 Mixed integer program

To provide an optimal comparison point for small problem sizes, and to further formalize our thinking about the problem, we formulated a mixed integer programming (MIP) solution. This section summarizes the MIP formulation (referred to as Optimal in subsequent sections); a more detailed description can be found in Appendix A.

The inputs to the MIP formulation are a set of penalty rates (e.g., outage, recent data loss and vulnerability penalty rates) for each application, device resource capabilities, and a recovery graph describing alternate recovery paths for each workload, including their jobs, resource requirements and precedence relationships. The MIP chooses a recovery path for

each workload and determines a schedule for the recovery operations, specified as the start time for each chosen job. The objective function is to minimize the overall penalties incurred by all workloads in the system.

Constraints govern the choices that can be made. For instance, for each application, only a single recovery path can be chosen, and all jobs on a workload's chosen path will be selected for execution. The chosen schedule must satisfy the precedence constraints specified in the input recovery graph. Constraints also govern resource usage: the sum of all resource demands for a given device must not exceed the capabilities for that device.

Our prototype MIP implementation uses ILOG's CPLEX solver [16], the standard tool for math programming search space exploration, to find an optimal solution.

3.3 Ad hoc heuristics

In addition to the Administrator and Optimal algorithms, we have implemented three ad hoc heuristics. Each heuristic has two steps: choosing a path and scheduling the jobs in the chosen paths. The algorithms use various combinations of priority-based, randomized, and genetic algorithm-guided decision-making for these two steps.

3.3.1 MLP-PRS

The first algorithm, called *min-loss path, penalty rate schedule* (MLP-PRS for short), deterministically picks the "best" path for each workload by choosing the one that minimizes recent data loss. If multiple paths are equivalent, the algorithm breaks ties by choosing the path that minimizes the data vulnerability penalties. (We note that the algorithm cannot use the actual outage times as the priority, because these times aren't known until the schedule is determined.) It then deterministically schedules jobs for the chosen paths from the list of eligible jobs in the order of the workload's outage penalty rate, as long as resource capacity constraints are not violated. Thus, eligible jobs from "high priority" workloads are preferentially scheduled over eligible jobs from "low priority" workloads. This heuristic is intended to be a more sophisticated version of the Administrator heuristic described in Section 3.1.

3.3.2 RP-PRS

In a resource-constrained environment, a greedy choice of paths may not always provide the best solution. The second heuristic, *random path, penalty rate schedule* (RP-PRS for short), randomly chooses the path for each workload, and uses the same penalty rate priority scheme as MLP-PRS to schedule eligible jobs. Path selection and scheduling are repeatedly applied until an execution time limit is reached, and the best alternative is chosen. This heuristic is intended to explore paths with non-minimal data loss that might be otherwise ignored by MLP-PRS.

We also explored an additional variant, *random path, random first fit schedule* (RP-FFS), which randomly chooses the path for each workload, and randomly schedules jobs from the list of eligible jobs, as long as the resource constraints are satisfied. For the examples we considered, we found that the RP-PRS algorithm outperformed the RP-FFS algorithm, so we discuss only the RP-PRS variant.

3.3.3 GAP-PRS

Randomization may not be the most efficient way to explore

the space of potential path selections. The third heuristic, *genetic algorithm path, penalty rate schedule* (GAP-PRS for short), uses a genetic algorithm to choose the path for each workload, and then uses the same penalty rate priority scheme as the previous two heuristics to schedule eligible jobs. This heuristic is intended to explore whether evolutionary adaptation can more efficiently explore the path space than a random search.

The genetic algorithm [26] considers a set of candidate solutions that evolves over time based on a set of adaptations, as described below. The algorithm starts with a population of 100 randomly chosen solutions (e.g., *individuals*). Using a genetic metaphor, the individual is defined by a set of *genes* that corresponds roughly to a vector of decision variables from the MIP formulation. Our individual is defined as a set of path choices, one per workload.

During each iteration of the algorithm, a new population is chosen by evaluating which candidate solutions are best using an evaluation (*fitness*) function analogous to the objective function in the MIP. The schedule for the evaluation is determined by applying the greedy penalty rate scheduling algorithm described above. In particular, the algorithm picks two individuals at random and applies the fitness function to determine the winner, which is included in the population for the next iteration. This process is repeated until the new population is filled. An individual in the old population may be selected multiple times, and therefore multiple copies of a good individual may appear in the new population. A place is also reserved in the new population for the best individual ever found.

The new population then undergoes several adaptations, including *crossover* and *mutation*. Each crossover operation randomly selects a pair of individuals and "mates" them with a certain probability (empirically set to 0.87 in our experiments) to produce two new individuals, which replace the parent individuals. (If no mating occurs, the new pair of individuals is the same as the original pair.) Mating or crossing two individuals entails swapping a subsequence of the genes, as defined by pivot points, which divide the genes into contiguous regions. Our crossover algorithm uses two pivot points, exchanging the genes between these pivot points, and retaining the original genes elsewhere. After crossover, genes are mutated with a certain probability (empirically set to 0.04 per gene in our experiments). Each gene mutates within its domain (e.g., [0, pathsInWkld] for path choice genes).

We experimented with a variant called *genetic algorithm path, genetic algorithm schedule* (GAP-GAS), that uses two distinct genetic algorithms: one to choose the path, and a second to determine the schedule. For the examples we considered, we found that the GAP-PRS and GAP-GAS algorithms performed equally well, so we discuss only the GAP-PRS variant.

We also tried to improve the results produced by the ad hoc algorithms (e.g., Administrator, MLP-PRS, etc.) by using a genetic algorithm that simultaneously evolves the path and the schedule, and seeds its initial population with the best solutions found by the ad hoc algorithms. In practice, we found that this algorithm didn't improve on the solutions found by the other algorithms, primarily because the other algorithms produced high-quality solutions at the outset. As a result, we do not present results for this algorithm in Section 4.

Type	Outage penalty rate	Recent loss penalty rate	Vulnerability penalty rate
<i>Student accounts (S)</i> : Storage of data owned by students is tolerant of recent data loss, outages and vulnerability.	\$0.5K	\$0.5K	\$0.5K
<i>Company documents (D)</i> : Documents such as presentations and design documents are tolerant to small outages and vulnerability, but less tolerant to loss of recent writes.	\$5K	\$500K	\$5K
<i>Web server for online retailer (W)</i> : Outages are expensive, because when the server is down, orders stop. Recent data loss and vulnerability are tolerable, because data can be replaced from other sources.	\$500K	\$5K	\$5K
<i>Consumer banking (C)</i> : Consumer banking is tolerant of modest outages (since account holders are unlikely to switch banks), but not data loss or vulnerability to additional failures.	\$50K	\$5M	\$50K
<i>Central banking (B)</i> : Central banks are required by regulations to have zero data loss and small outage windows; they also desire negligible vulnerability.	\$5M	\$5M	\$50K

Table 1: Summary of requirements for example workloads. These workloads and penalty rates are based on the industry segment examples described in [21].

4. EXPERIMENTAL RESULTS

In this section, we present experimental results to illustrate the recovery solver’s operation. We begin with two simple case studies of small-scale environments, to build the reader’s intuition. We then study the scalability of our algorithms using larger configurations. Finally, we examine the sensitivity of the algorithms to execution time, input information, and the tightness of resource constraints.

4.1 Environment

We assume that all workloads are protected using the combination of techniques illustrated in Figures 1 and 2, including synchronous mirroring, intra-array split mirroring, tape backup and tape vaulting. Recovery from array failure or site disaster can be achieved through failover to the remote site or reconstruction of the mirrored or vaulted copy to the target primary site. Our case study environments employ disk arrays, tape libraries and inter-site links. Table 5 in Appendix B describes the bandwidth and capacity capabilities of these device types.

Our case studies use five different application classes whose penalty rate magnitudes are based on market research data [23], as described in Table 1. For simplicity, we assume that the capacity and access characteristics for all applications are the same. They are based on the access characteristics for the cello2002 workload described in [21]. Table 6 in Appendix B describes these characteristics for normal mode (e.g., non-degraded) operation.

4.2 Simple case studies

In this section, we explore several simple case studies to build intuition. These studies model a primary-secondary site environment and a two-way peer environment.

4.2.1 Case study 1: primary-secondary sites

The primary-secondary scenario is intended to model a shared disaster recovery site environment, where workloads

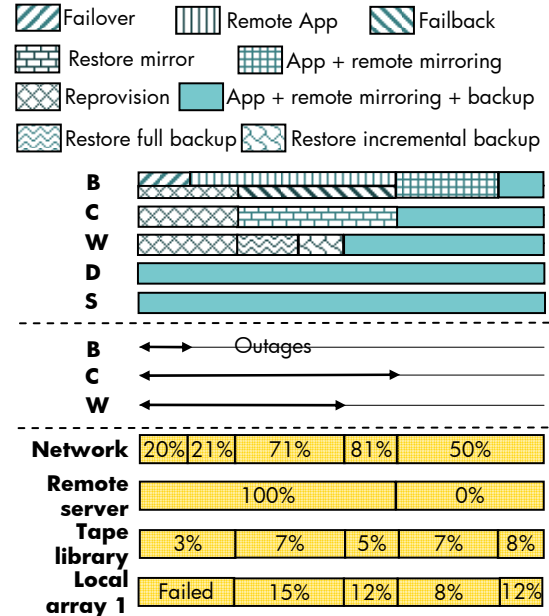


Figure 3: Evaluation of optimal solution to primary-secondary scenario. This figure shows a scheduling chart, data outage periods and resource utilization.

running at a primary site are protected by mirroring and vaulting to the recovery site. The shared nature means that customers pay a reduced fee to share access to the recovery site resources, as compared to a dedicated facility, where resources are dedicated to a particular workload, albeit at a higher fee. Customers gain access to recovery site resources by invoking a disaster, and resources are consumed on a first-come-first-served basis.

In this scenario (illustrated in Figure 1), the primary copies for five workloads are maintained at the first site, with workloads *B*, *C* and *W* on one array and workloads *D* and *S* on a second array. Each primary workload uses a dedicated server at the first site. The recovery site uses two arrays to mirror these workloads, and has a single server. We examine recovery after the failure of the primary array hosting workloads *B*, *C* and *W*.

Figure 3 illustrates an optimal schedule, and Table 2 summarizes the optimal path choices and the resulting dependency properties and penalties. In the absence of contention from other workloads, the workloads with the highest outage penalty rates (*B*, *W*, and *C*) would failover to the recovery site’s mirror copy, to minimize their downtime and data loss. However, due to the limited resources at the recovery site, when multiple workloads are present, failovers must be serialized. As a result, in the context of multiple workloads, the optimal schedule reconstructs *C* from its mirror copy, instead of waiting to failover until after *B* has completed its failover and failback. Workload *W* has both a high outage penalty rate and a low data loss penalty rate, so its optimal recovery path is to restore from a local backup copy, trading off a small amount of recent data loss for a shorter recovery time than restoring from the remote mirror, or waiting to failover.

Metric	B	C	W	D	S
Expected path (in isolation)	FM	FM	FM	n/a	n/a
Optimal path (in context)	FM	RM	RB	n/a	n/a
Outage time (hr)	1	31	24	0	0
Loss time (hr)	0	0	48	0	0
Vulnerability time (hr)	30	0	0	0	0
Per-workload penalty (M \$)	5.0	1.55	12.24	0.00	0.00

Table 2: Summary of optimal primary-secondary solution. The expected path row describes the path that one would expect to be chosen, if this workload were operating in isolation. The optimal path row describes the optimal path choice when multiple workloads are present, due to contention for resources. Recall that FM means failover to mirror, RM means restore from mirror, and RB means restore from backup.

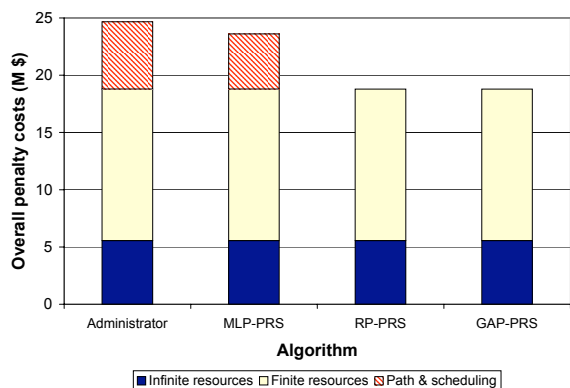


Figure 4: Comparison of algorithm solution quality for primary-secondary scenario.

Figure 4 compares the quality of the solutions provided by the various algorithms by breaking down the overall penalties into three categories. *Infinite resources penalties* are the penalties due to the duration of recovery tasks and the existence of precedence constraints; they represent the penalties for the preferred path and schedule choices under infinite resources, where contention is not an issue. *Finite resources penalties* are the penalties incurred due to the scheduling compromises that must be made to address resource contention in a finite resource environment; the sum of finite and infinite resource penalties is equivalent to the optimal solution. Finally, *path and scheduling penalties* are the penalties due to inefficiencies in an algorithm's recovery path selection or recovery scheduling. Infinite resources penalties are inherent penalties that cannot be eliminated, whereas path and scheduling and finite resources penalties can be addressed through algorithm design and overprovisioning of resources, respectively. As a result, we compare algorithms on the basis of the percent reduction in these addressable costs, relative to the Administrator algorithm.

We observe that the RP-PRS and GAP-PRS algorithms both produce optimal solutions, resulting in a 44% (or \$5.9M) improvement over the Administrator algorithm. The

Metric	C1	C2	C3	W	D
Expected path (in isolation)	FM	FM	FM	n/a	n/a
Optimal path (in context)	FM	RM	RM	n/a	n/a
Outage time (hr)	1	31	31	0	31
Loss time (hr)	0	0	0	0	0
Vulnerability time (hr)	30	0	0	0	0
Per-workload penalty (M \$)	0.05	1.55	1.55	0.00	0.16

Table 3: Summary of optimal two-way peer solution.

MLP-PRS algorithm results in a much more limited (e.g., 6%) improvement over Administrator. The MLP-PRS and Administrator algorithms do not achieve optimal solutions because they do not consider paths that lead to non-minimal data loss, such as recovering from a backup copy.

4.2.2 Case study 2: peer sites

The peer scenario is intended to model an environment where two sites each serve as the primary for a subset of the workloads, and as a secondary (i.e., remote mirror) for the workloads served primarily by the other site. This might occur, for example, in a corporation with multiple branch offices that serve as peers for one another.

In this example, the first site serves as primary for three consumer banking workloads (*C1*, *C2* and *C3*), and the second site serves as primary for two workloads (*W* and *D*). At each site, one array hosts *C1*, *C2* and *C3*, and another array hosts *W* and *D*. We assume that each site is provisioned with enough servers for its primary workloads. We examine recovery after the failure of the primary array at the first site hosting workloads *C1*, *C2* and *C3*.

Table 3 summarizes the optimal path choices and the resulting dependability properties and penalties. As expected, the highest penalty rate workloads (*W*, which was unaffected by the failure, and *C1*, one of the failed consumer banking workloads) are scheduled first to use the server resources at site2 to minimize their outage time. The remaining failed workloads (*C2* and *C3*) recover from the remote mirror copy, rather than waiting to serialize their failover after *C1*'s failover and fail-back. Once *C1* has completed failback, the server resources at site2 become available for restarting workload *D*.

Figure 5 illustrates the effectiveness of the various algorithms. RP-PRS and GAP-PRS achieve optimal solutions, resulting in a 39% (or \$1.2M) improvement in addressable penalty costs over the Administrator solution. MLP-PRS results in no improvement over Administrator. These algorithms always pick the minimal data loss path, and in the case where multiple paths have the same data loss, they choose the path that would produce the shortest outage in the absence of resource contention. This policy leads to the choice of failover for all three failed workloads. Tasks are scheduled according to descending outage penalty rate order, meaning that the unaffected *W* will be scheduled immediately, forcing the serialization of the failover operations, and leading to a sub-optimal schedule.

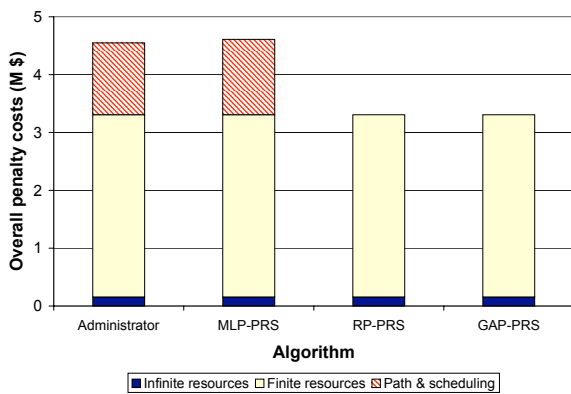


Figure 5: Comparison of algorithm solution quality for peer scenario.

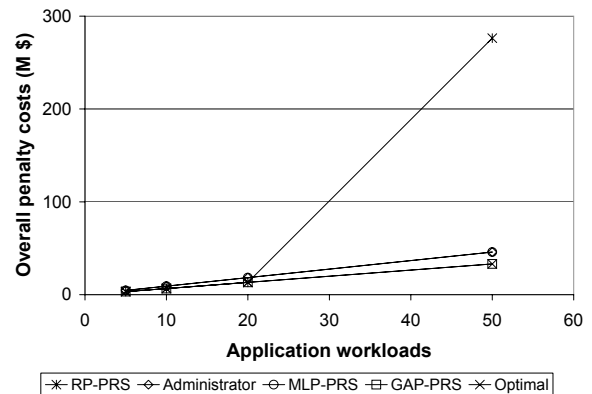


Figure 7: Scalability of solution quality for separate administrative domains peer scenario.

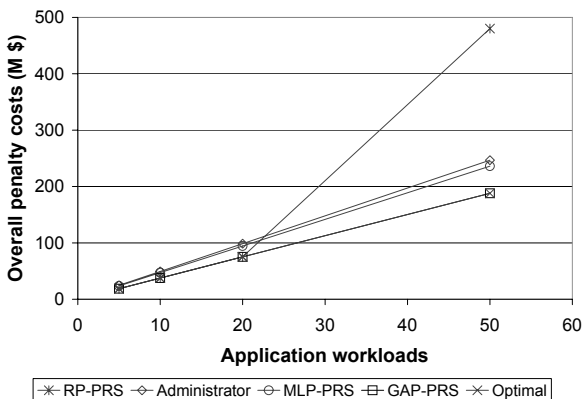


Figure 6: Scalability of solution quality for separate administrative domains primary-secondary scenario.

4.3 Scalability experiments

Although the examples in the previous section are useful for building intuition about recovery schedules, many real-world environments have more applications – up to tens of applications, according to solution architects. In this section, we explore the scalability of our formulations as the size of the environment increases.

4.3.1 Separate administrative domains

We begin with an environment that models a service provider that maintains separate administrative domains for each of its customers. This environment is constructed from the smaller environments used in Section 4.2; both the collection of workloads and the devices are replicated. We assume a failure model of one array per administrative domain, analogous to the failures used in Section 4.2. The *scale factor* describes how many administrative domains (e.g., copies of the smaller environment) are used in a given experiment.

The benefit of these experiments is that the optimal solution is known. Section 4.2 presents the MIP's optimal objective function for the smallest (e.g., scale factor SF1) experiment. The MIP formulation solver is unable to complete execution

for problems with scale factor larger than SF1 (it runs out of memory); however, we can derive the optimal solutions for these larger problems from the solutions to the SF1 experiments. Because the individual administrative domains operate independently, the optimal objective function as the scale factor grows is merely the scale factor times the baseline (SF1) objective function.

To provide a meaningful comparison between the algorithms, we bound the amount of execution time given to each algorithm for a given problem size. Our experimental platform is an HP ProLiant DL360G3 dual-processor server with 2.8 GHz Intel Xeon processors and 4 GB of main memory, running Debian Linux version 2.4.27. The execution time bounds are: one minute for five application workloads, five minutes for ten workloads, ten minutes for twenty workloads and one hour for fifty workloads. We note that Administrator and MLP-PRS finish significantly before these bounds (e.g., in well under one second for all problem sizes), because they only consider a single alternative, whereas the other algorithms use the entire execution time allotment.

Figure 6 illustrates algorithm scalability for the primary-secondary scenario. MLP-PRS provides a modest (e.g., 6%), but consistent improvement over Administrator for all problem sizes. The RP-PRS and GAP-PRS algorithms produce optimal solutions for small to medium problem sizes, thus providing up to 44% improvement in addressable costs over solutions produced by Administrator. Solution quality degrades for RP-PRS for the largest problem size, though, because it is unable to explore a sufficiently large fraction of the possible solution space. This solution space increases exponentially as the number of workloads increases linearly. For the 50-workload problem, GAP-PRS successfully uses the genetic algorithm to select the optimal paths, and the priority-based algorithm to determine an optimal schedule, resulting in a 44% (or \$58.6M) improvement in addressable costs over Administrator.

For the peer scenario (shown in Figure 7), MLP-PRS isn't able to improve on the solutions provided by Administrator, due to their path and scheduling decisions, as outlined in Section 4.2. RP-PRS and GAP-PRS achieve optimal solutions for small and medium problem sizes within the

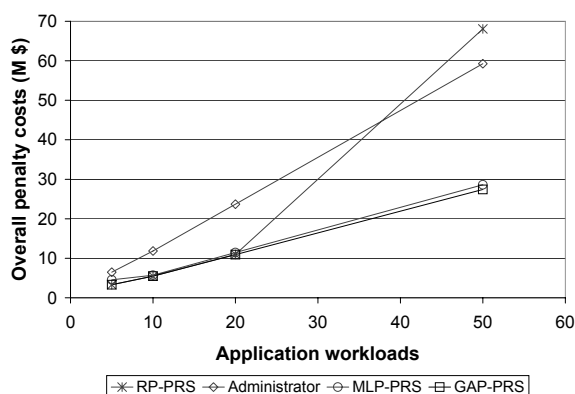


Figure 8: Scalability of solution quality for site consolidation peer scenario.

execution time bounds, resulting in a 41% improvement over Administrator. However, as with the primary-secondary scenario, solution quality for RP-PRS degrades for the largest problem size, given the prescribed execution time. Again, GAP-PRS finds optimal solutions and a 41% (\$12.8M) improvement over Administrator, even for large problem sizes.

4.3.2 Site consolidation

We also explored a site consolidation environment using a single aggregate resource per resource type (e.g., server, disk array or tape library). These consolidation experiments are intended to show that our algorithms do not rely on knowledge of the problem structure to determine high-quality solutions. We found that the primary-secondary scenario results are similar to the separate administrative domains case, so we focus our discussion on the peer scenario.

The experiments scale the capabilities of the aggregate resources, so that sufficient bandwidth exists to run all of the workloads. Each primary site has a large enough aggregate server to run its primary workloads. In each peer scenario, 60% of the workloads are consumer banking, 20% are web server, and 20% are company documents. (This application mix is the same as the separate administrative domains case.) As in the separate administrative domains case, for five workloads, the first site acts as primary for the three consumer banking workloads, and the second site serves as primary for the web server and company documents workloads. This base case differs from the one outlined in Section 4.3.1, however, in that all workloads are affected in some way by the failure: workloads whose primary copy was at the failed site need to be recovered, and those whose secondary copy was at the failed site must wait until that site has been reprovisioned before resuming remote mirror operation. In the larger scenarios, each site serves as the primary for half of the workloads in each application category.

For these experiments, the optimal solution is unknown, as our MIP solution does not scale beyond the five workload case and, unlike the separate administrative domains scenario, the optimal solution cannot be derived from the solution of a smaller problem. Because we cannot determine the optimal solution for the larger problem sizes, we cannot distinguish

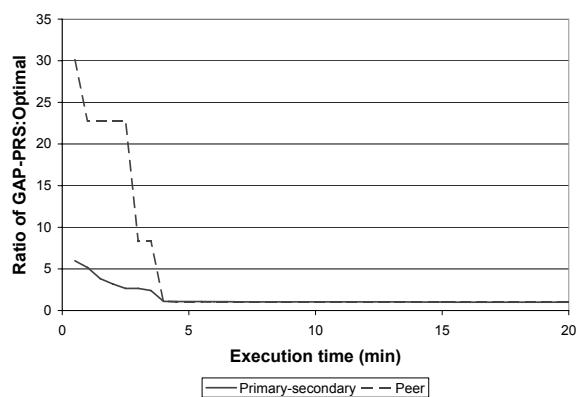


Figure 9: Execution time sensitivity for GAP-PRS for the 50-application separate administrative domains scenarios.

between the finite resources penalties and path and scheduling penalties, as in Figure 5. As before, we compare algorithms on the basis of the percent reduction in overall (e.g., finite resources and path and scheduling) addressable costs, relative to the Administrator algorithm.

Figure 8 illustrates algorithm scalability for the site consolidation peer scenario. The Administrator algorithm consistently performs worse than the other algorithms at all problem sizes, due to its coarse-grained scheduling policy, which under-utilizes the available device resources. In the five-workload case, MLP-PRS provides 1.4X improvement in overall addressable costs, and both RP-PRS and GAP-PRS provide 2.0X improvement. For medium-sized problems (e.g., up to 20 applications), MLP-PRS improves on the Administrator solution by 2.2X, and both RP-PRS and GAP-PRS improve by 2.3X. These trends continue through 50 applications (resulting in savings of up to \$31.8M) for all of the algorithms except RP-PRS, which cannot sufficiently explore the large solution space in the bounded one-hour execution time.

4.4 Sensitivity experiments

In this section, we examine the sensitivity of our results to algorithm execution time, input data specification and the tightness of resource constraints.

4.4.1 Execution time

The bounded execution times used for the scalability results in the previous section were sufficient for the GAP-PRS algorithm to achieve optimal results, even for large problem sizes. These bounds (e.g., one hour for the 50-application case) may seem long, however, if an administrator wants to start recovery as soon as possible after an incident. In this section, we explore whether the GAP-PRS algorithm might be used more interactively, even for large recovery schedules.

Figure 9 illustrates how quickly the GAP-PRS algorithm converges, by plotting the ratio of solution objective functions for GAP-PRS and Optimal. Results are shown for the 50-application primary-secondary and peer variants of the separate administrative domains environment. (Recall that we can extrapolate the optimal solution objective function for the separate administrative domains environment, even for large

problem sizes.) In both cases, GAP-PRS's solutions converge to within 10% of optimal within the first five minutes of execution. Solutions within 1% of optimal are achieved within 13 minutes (peer scenario) and within 16 minutes (primary-secondary), respectively. Thus, it is possible to achieve near-optimal solutions in considerably less execution time.

4.4.2 Input information

A common criticism of automated approaches is that they require too much input data, especially high-level requirements (e.g., penalty rates), which may be hard to quantify. We investigate the sensitivity of our solutions to the choice of penalty rates by simulating the case where the administrator is only able to provide inexact penalty rates. We compare the solution chosen in this case to the solution chosen if the penalty rates are more precisely specified. More specifically, we evaluate the objective function for the solution chosen with the inexact penalty rates using the precise penalty rates, and compare this result with the objective function for the solution chosen with the precise penalty rates.

Here we present results for the primary-secondary scenario, with the mixture of workloads described in Section 4.2.1. We simulate inexact penalty rates by converting all penalty rates into an order-of-magnitude estimate (e.g., \$100/hr, \$1000/hr, etc.) Precise penalty rates are represented by a range of values at each order of magnitude (e.g., \$10K/hr, \$20K/hr, \$30K/hr, etc., for the consumer banking workloads). We found that the objective functions for the solutions provided by inexact and precise inputs differed by less than one percent, for all ranges of workloads (e.g., five through fifty workloads). We found this result to be true for both the separate administrative domain environment, as well as the site consolidation environment.

Although the precise specification of input requirements may be unnecessary, our approach does require the ability to differentiate between different classes of applications. Several techniques may assist administrators in quantifying their requirements [20]. First, the best practices in the consulting community include *business impact assessment (BIA)* interviews and questionnaires, to assess the financial impacts of downtime. Second, marketing research firms can provide rules of thumb for particular industry segments (e.g., [23]). Third, as described in [18], even if users can't quantify their requirements, we may be able to provide toolsets to allow them to perform "what if" analyses to explore the design ramifications of different input requirements. Finally, knowledge of higher-level business processes and their data flows may provide direct insight into data requirements.

4.4.3 Resource constraint tightness

As shown in our earlier experiments, contention for resources during recovery may lead to non-obvious recovery path or scheduling decisions. Resource overprovisioning, which is often used to address many other IT concerns, may permit simpler recovery scheduling algorithms to find optimal or near-optimal solutions.

We re-evaluated the separate administrative domains primary-secondary and peer environments from Section 4.3.1, but increased the resource capabilities by 10X, thus removing any resource contention. We observe that all algorithms (MLP-PRS, RP-PRS and GAP-PRS) find optimal solutions for small and medium-sized problems. MLP-PRS

and GAP-PRS also find an optimal solution for the largest problem size. The heuristic used by Administrator consistently finds solutions that are 4.4X worse than optimal, due to the forced serialization of workloads in different priority classes, even though sufficient resources exist for inter-class parallelism.

Although overprovisioning does simplify the solution of the recovery scheduling problem, it results in grossly under-utilized resources under normal operation. For instance, servers purchased or leased for the sole purpose of recovery may remain idle during normal operation. Ultimately, the question of overprovisioning is a pre-disaster design question. System designers should balance the additional outlay costs of overprovisioning against the potential benefits in penalty reduction for recovery after a disaster. Keeton, et al., describe one method for exploring these tradeoffs [21]. The recovery scheduling solution techniques described in this paper can be used to provide detailed recovery time and data loss estimates, to better inform this design process.

4.5 Discussion

The coarse-grained categorical recovery scheduling done by the Administrator algorithm does not produce optimal schedules, indicating that there is opportunity for improvement beyond what human administrators can do today. We can achieve better solutions if we are willing to introduce more parallelism between categories, as in the MLP-PRS algorithm. However, this scheduling parallelism is difficult for humans to reason about as the number of workloads increases, so it is unlikely to be adopted in practice, without support for automation.

If resource constraints are loose, then MLP-PRS's combination of greedy path and scheduling decisions will likely produce the optimal schedule. However, if resource constraints are tight, then greedy decisions (like those made by both Administrator and MLP-PRS) about recovery paths are unlikely to be optimal. For instance, it may be better to recover from a backup copy than to wait for the opportunity to failover to a remote site. The monetary penalty for these suboptimal schedules depends on the penalty rates of the workloads, but can easily scale to millions of dollars for even small problem sizes. Ultimately, the question of resource overprovisioning to speed recovery is a design tradeoff between up-front outlay costs and post-disaster recovery penalties.

One technique to overcome sub-optimal decisions for tighter resource environments is to introduce randomization, such as through the RP-PRS algorithm. This approach does well if the algorithm can explore a sufficiently large portion of the solution space. However, as the number of workloads increases linearly, the size of the potential solution space increases exponentially, and the quality of the solution in bounded time is poor. Good answers are possible if the algorithm is allowed to run sufficiently long (e.g., in an offline fashion). The most effective approach we observed is to use evolutionary pressure to pick the best path, along with a greedy scheduling algorithm. We've demonstrated that this approach provides optimal results in bounded time, and converges to near-optimal solutions in even less time, meaning that recovery schedules could easily be produced on the fly.

For our case studies, the penalty rate scheduling provides a

good schedule, once the recovery path is chosen. We note that there may be other scenarios where this greedy approach to scheduling doesn't provide the best answer. We are exploring these scenarios as part of future work.

5. RELATED WORK

Practitioners' guides (e.g., [8, 24, 25, 33]) offer rules of thumb and high-level guidance for designing dependable storage systems, and recovering these systems after disasters. These books focus mainly on the logistical, organizational and human aspects of disaster recovery, such as the need for a documented disaster recovery plan, the information it should contain, the need to maintain a list of critical personnel and their telephone numbers at a redundant location, etc. They do not treat detailed scheduling or resource allocation issues.

Workflow management [9, 31] provides abstractions for describing processes and their constituent activities, as well as relationships between activities, criteria for starting and terminating processes, and information about individual activities (e.g., participants, associated applications and data). This area of research provides many alternative languages and management tools, which may be useful for expressing and manipulating recovery graphs.

Several recent studies explore how to design a storage system to meet user-defined goals. Total Recall [5] is a peer-to-peer storage system that controls the level of redundancy in the system to meet a user-specified availability goal. Keeton, et al., present methods for automatically selecting data protection techniques (such as remote mirroring or tape backup) for storage systems to minimize overall cost of the system, including up-front outlay costs and recovery penalties [21]. From that paper, we have adopted the basic approach of using data loss and data outage penalties to evaluate alternative solutions. Keeton and Merchant present a model for the recovery time and data loss after a storage system failure [19]. The previous work considers only a single recovery workload, and hence does not investigate the question of scheduling competing recovery tasks. In contrast, this paper focuses on scheduling recovery workloads to minimize the costs of a failure affecting multiple applications.

In the operations research scheduling literature, our treatment of data recovery scheduling problem is most closely related to project scheduling with resource constraints [28]. In a project scheduling problem, jobs are subject to precedence constraints and the objective is to minimize makespan, or the duration of the overall schedule. The setting is a parallel environment with constrained resources. Authors have proposed several methods to tackle this problem, including the critical path method and the PERT method [3, 4, 6], genetic algorithms [11, 32], and other heuristics [22]. The scheduling problem has also been addressed in other contexts, including animation rendering [1], real-time systems [36], embedded systems [10] and multiprocessors [15, 35]. The data recovery scheduling problem differs from the ones addressed in the literature in that it requires a choice between multiple alternative plans to complete the project; typically in project scheduling there is just one way to accomplish a project. Whereas the related literature generally minimizes makespan as its objective, our problem's objective function uses penalty rates that depend on the chosen project plan and on the state of completion of the chosen project plan. In addition, the job duration

depends on the amount of resources assigned to the job.

6. CONCLUSIONS

The data recovery scheduling problem is challenging: with multiple workloads to recover and multiple strategies for each workload's recovery, the number of possible schedules is large. Unfortunately, the categorical approaches administrators use to devise recovery plans don't provide optimal solutions: their inefficiencies result in millions of dollars of extra penalties in the face of disasters.

This paper makes several contributions toward the solution of the data recovery scheduling problem. Our approach of describing the recovery process as a recovery graph provides structure to an otherwise ill-defined process. This approach is useful for improving information and feedback gathering activities, rehearsing recovery plans and evaluating these drills.

Given the recovery graph abstraction, we formulate the data recovery scheduling problem as an optimization problem. We present several formulations to solving the data recovery scheduling problem, including priority-based, randomized, and genetic algorithm-guided ad hoc heuristics. Although simple, greedy approaches to path selection (such as MLP-PRS) may be effective if resource constraints are loose, these approaches don't succeed in more tightly constrained environments. Randomization (e.g., RP-PRS) helps here, if the algorithm is permitted to search a sufficiently large portion of the solution space. The best overall performance is provided by the GAP-PRS algorithm, which uses a genetic algorithm to choose paths and a greedy approach for determining the schedule. GAP-PRS achieves this goal in a bounded amount of time that could be used to generate recovery schedules interactively post-disaster, so that they can be tailored to the disaster at hand, thus increasing the likelihood of successful recovery.

We believe that this work is an important first step towards tackling this rich problem area. Many open questions remain, however, including extending the formulations to multiple resource demand attributes, coping with failures during the recovery process, incorporating application-level recovery and dependencies in our recovery scheduling, and incorporating measurements of real recoveries into our models.

Acknowledgements

The authors thank Caleb Lizarraga for implementing and evaluating several variants of the genetic algorithms described in Section 3.3. We thank Pedro Flores from the University of Sonora and Evan Kirshenbaum from HP Labs for discussions that helped shape our thinking on genetic algorithms. We thank Armando Fox, Terence Kelly, Ken Salem, Janet Wiener and John Wilkes, our shepherd Ant Rowstron, and the anonymous reviewers for their helpful comments on earlier versions of this work.

7. REFERENCES

- [1] E. Anderson, D. Beyer, K. Chaudhuri, T. Kelly, N. Salazar, C. Santos, R. Swaminathan, R. Tarjan, J. Wiener, and Y. Zhou. Value-maximizing deadline scheduling and its application to animation rendering. In

- Proc. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, July 2005.
- [2] A. Azagury, M. Factor, and J. Satran. Point-in-Time copy: yesterday, today and tomorrow. In *Proc. 10th NASA Conf. on Mass Storage Systems and Technologies/19th IEEE Symp. on Mass Storage Systems*, pages 259–270, April 2002.
- [3] K. R. Baker. *Introduction to sequencing and scheduling*. John Wiley, 1974.
- [4] E. Balas. Project scheduling with resource constraints. In E. Beale, editor, *Applications of Mathematical Programming Techniques*, pages 187–200. American Elsevier, 1970.
- [5] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. Total Recall: system support for automated availability management. In *Proc. ACM/USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, March 2004.
- [6] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource constrained project scheduling: notation, classification, models, and methods. *European Journal of Operations Research*, 112:3–41, 1999.
- [7] A. Chervenak, V. Vellanki, and Z. Kurmas. Protecting file systems: a survey of backup techniques. In *Proc. 6th NASA Conf. on Mass Storage Systems and Technologies/15th IEEE Symp. on Mass Storage Systems*, March 1998.
- [8] D. Cougias, E. Heiberger, and K. Koop. *The backup book: disaster recovery from desktop to data center*. Schaser-Vartan Books, Lecanto, FL, 2003.
- [9] P. de Jong. Going with the flow. *ACM Queue*, pages 25–32, March 2006.
- [10] C. Ekelin. *An optimization framework for scheduling of embedded real-time systems*. PhD thesis, Chalmers University of Technology, 2004.
- [11] S. Hartmann. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, 49:433–448, 1001.
- [12] Hewlett-Packard Company. *HP StorageWorks Enterprise Virtual Array*, December 2003. h18006.www1.hp.com/products/storageworks/enterprise/.
- [13] Hewlett Packard Company. *HP StorageWorks Extended Tape Library Architecture*, December 2003. h18006.www1.hp.com/products/storageworks/tlarchitecture/.
- [14] Hewlett-Packard Development Co. *HP OpenView Storage Data Protector administrator's guide*, October 2004. Mfg. Part Number B6960-90106, Release A.05.50.
- [15] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel and Distributed Systems*, 5(2):113–120, 1994.
- [16] ILOG, Inc., Mountain View, CA. *CPLEX 8.0 User's Manual*, July 2002. Available from <http://www.ilog.com>.
- [17] M. Ji, A. Veitch, and J. Wilkes. Seneca: remote mirroring done write. In *Proc. USENIX Annual Technical Conf.*, pages 253–268, June 2003.
- [18] K. Keeton, D. Beyer, J. Chase, C. Santos, and J. Wilkes. Lessons and challenges in automating data dependability. In *Proc. 11th ACM-SIGOPS European Workshop*, September 2004.
- [19] K. Keeton and A. Merchant. A framework for evaluating storage system dependability. In *Proc. Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 877–886, 2004.
- [20] K. Keeton and A. Merchant. Challenges in managing dependable data systems. *ACM SIGMETRICS Performance Evaluation Review*, March 2006.
- [21] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. Designing for disasters. In *Proc. USENIX Conf. on File and Storage Technologies (FAST)*, pages 59–72, March 2004.
- [22] R. Kolisch and S. Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In J. Weglarz, editor, *Project scheduling: recent models, algorithms and applications*, pages 147–178. Kluwer Academic Publishers, 1999.
- [23] Eagle Rock Alliance Ltd. Online survey results: 2001 cost of downtime. http://contingencyplanningresearch.com/2001_Survey.pdf, August 2001.
- [24] E. Marcus and H. Stern. *Blueprints for high availability*. Wiley Publishing, Indianapolis, IN, 2003.
- [25] P. Massiglia and E. Marcus, editors. *The resilient enterprise: recovering information services from disaster*. Veritas Software Corp., Mountain View, CA, USA, 2002.
- [26] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, third edition, 1999.
- [27] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. SIGMOD*, pages 109–16, 1–3 June 1988.
- [28] M. Pinedo. *Planning and scheduling in manufacturing and services*. Springer Series in Operations Research. Springer-Verlag, 2005.
- [29] Y. Saito, S. Frolund, A. Veitch, A. Merchant, and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. In *Proc. ACM Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 48–58, October 2004.
- [30] R.R. Schulman. *Disaster recovery issues and solutions*. Hitachi Data Systems White paper, September 2004.
- [31] W. van der Aalst and K. van Hee. *Workflow management: models, methods and systems*. MIT Press, Cambridge, MA, USA, 2002.
- [32] M. Wall. *A genetic algorithm for resource-constrained scheduling*. PhD thesis, Massachusetts Institute of Technology, June 1996.
- [33] C. Warrick et al. *IBM TotalStorage business continuity solutions guide*. IBM Redbooks. IBM International Technical Support Organization, August 2005.
- [34] J. Wylie, M. Bigrigg, J. Strunk, G. Ganger, H. Kiliççöte, and P. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, August 2000.
- [35] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Softw. Eng.*, 19(2):139–154,

OBJECTIVE FUNCTION

$$\text{Minimize : } \sum_j \pi_{\text{out},j} \cdot \sum_t t \cdot s_{j,t} + \sum_w \sum_p \pi_{\text{loss},p} \cdot z_{w,p} + \sum_j \pi_{\text{vul},j} \cdot \left(\sum_t t \cdot s_{S(j),t} - \sum_t t \cdot s_{j,t} \right) \quad (1)$$

CONSTRAINTS

$$\sum_p z_{w,p} \cdot V_{w,p} = 1 \quad \forall w \quad (2)$$

$$y_j = \sum_w \sum_p Q_{p,j} \cdot z_{w,p} \cdot V_{w,p} \quad \forall j \quad (3)$$

$$\sum_t s_{j,t} = y_j \quad \forall j \quad (4)$$

$$\sum_t e_{j,t} = y_j \quad \forall j \quad (5)$$

$$\sum_{\tau=1}^t s_{j,\tau} \geq \sum_{\tau=1}^t e_{j,\tau} \quad \forall j, t \quad (6)$$

$$\sum_{\tau=1}^t s_{j,\tau} \leq \sum_{\tau=1}^t e_{i,\tau} \quad \forall j, i \in P(j), t \quad (7)$$

$$e_{j,\tau} = s_{j,\tau-D_j} \quad \forall j \in J_{\text{rec}}, t, \tau = t + D_j, t + D_j + 1, \dots, T \quad (8)$$

$$e_{i,t} = s_{j,t} \quad \forall i \in J_{\text{oper}}, j \in S(i), t \quad (9)$$

$$e_{j,t} = \begin{cases} y_j & t = T \\ 0 & \text{otherwise} \end{cases} \quad \forall j \in J_{\text{fin}}, t \quad (10)$$

$$r_{j,t} \geq \sum_{\tau=1}^t s_{j,\tau} + \sum_{\tau=t+1}^T e_{j,\tau} - 1 \quad \forall j, t = 1, 2, \dots, T-1 \quad (11)$$

$$r_{j,t} \leq \sum_{\tau=1}^t s_{j,\tau} \quad \forall j, t = 1, 2, \dots, T-1 \quad (12)$$

$$r_{j,t} \leq \sum_{\tau=t+1}^T e_{j,\tau} \quad \forall j, t = 1, 2, \dots, T-1 \quad (13)$$

$$\sum_j R_{j,m} \cdot r_{j,t} \leq K_{m,t} \quad \forall m, t \quad (14)$$

Figure 10: Discrete time mixed integer program (MIP) formulation of recovery scheduling problem.

1993.

- [36] W. Zhao, K. Ramamritham, and J. A. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Trans. on Software Engineering*, 13(5):564–577, 1987.
- [37] W.-D. Zhu et al. *IBM Content Manager backup/recovery and high availability: strategies, options and procedures*. IBM Redbook, March 2004.

APPENDIX

A. MIP FORMULATION

This appendix presents the details of our MIP formulation. Table 4 defines the variables, and Figure 10 lists the objective function and constraints.

A workload’s business requirements are represented by penalty rates for data outages ($\pi_{\text{out},j}$), data loss ($\pi_{\text{loss},p}$) and data vulnerability ($\pi_{\text{vul},j}$). The inputs also include a recovery graph, which consists of one or more recovery paths ($V_{w,p}$). Each path contains one or more jobs ($Q_{p,j}$), where the term “jobs” encompasses both recovery tasks and operational states. The graph represents precedence constraints ($I_{j,i}$) for when job i cannot begin until job j has completed. (Predecessor and successor jobs are further described by $P(j)$ and $S(j)$, respectively.)

For each task, $R_{j,m}$ indicates the resources required for that task (e.g., the minimum bandwidth requirements for operational states or a minimum expected bandwidth allocation for recovery tasks). Constraints will check to ensure that the sum of all resource demands for a given device doesn’t exceed the capabilities of that device (e.g., $K_{m,t}$). D_j describes the duration required for fixed work tasks. This duration may be a fixed delay (e.g., for provisioning resources or retrieving tapes from a remote vault), or it may be the time to perform work (e.g., to transfer incremental backup data from tape) at the requested bandwidth allocation level.

Figure 10 presents our discrete time mixed integer programming (MIP) formulation. The formulation’s objective is to determine the solution that minimizes the overall penalties incurred by all workloads, as quantified in Eq 1. This solution includes a recovery path choice for each workload (e.g., $z_{w,p}$), the choice of the jobs along that path (e.g., y_j), and a schedule of jobs (e.g., indicated by a set of task start time indicators $s_{j,t}$).

Constraints govern the choices that can be made. For each workload, only a single recovery path can be chosen, and the chosen path must be associated with that workload (Eq 2). All jobs on a workload’s chosen path will be selected for execution (Eq 3). Each job that’s chosen for execution starts (Eq 4) and ends (Eq 5) only once. Jobs must start before they end (Eq 6). The selected start times must satisfy the precedence constraints specified in the input recovery graph (Eq 7).

Constraints also govern resource usage. We use a job’s end time, as indicated by $e_{j,t}$, to track how long that job consumes resources. Jobs corresponding to recovery tasks consume resources until their work has been completed (e.g., for D_j , as shown in Eq 8). Jobs corresponding to operational states consume resources until the next job begins for “intermediate” tasks (Eq 9), and until the end of the schedule for the final job in a path (Eq 10).

A job holds resources after it starts and before it ends (Eq 11). It doesn’t hold resources before it starts (Eq 12) or after it ends (Eq 13). For each device, the resources consumed must not exceed the device’s capabilities (Eq 14).

B. ENVIRONMENT DETAILS

This appendix quantifies our experimental parameters. Table 5 quantifies device parameters, and Table 6 presents the application and data protection workload bandwidth demands in the normal mode of operation.

Parameter	Notation	Units	Description
<i>Indices</i>			
workloads	$w \in W$		set of workloads in the recovery graph
paths	$p \in P$		set of paths in the recovery graph (i.e., across all workloads)
jobs	$j, i \in J$		set of jobs (tasks) in the recovery graph (e.g., across all paths). Jobs are categorized as recovery tasks (J_{rec}), operational states (J_{oper}) or final (J_{fin}).
time slots	$t \in T$		set of fixed-duration time slots in the schedule
devices	$m \in M$		set of device (e.g., "machine") instances
<i>Input parameters</i>			
data outage penalty rate	$\pi_{out,j}$	US \$ / hour	outage penalty rate for job j . We observe the convention that this penalty rate is non-zero for the first job where an application is resumed, and zero otherwise.
data loss penalty	$\pi_{loss,p}$	US \$	penalty for recovery path p , which corresponds to the product of data loss penalty rate times the recent data loss for path p
data vulnerability penalty rate	$\pi_{vul,j}$	US \$ / hour	vulnerability penalty rate for job j . We observe the convention that this penalty rate is non-zero for jobs where an application is resumed, but its entire set of data protection workloads has not yet resumed; it is zero, otherwise.
recovery path validity	$V_{w,p}$		binary indicator of whether a recovery path p is associated with a workload w
recovery path membership	$Q_{p,j}$		binary indicator of whether a job j is associated with a recovery path p
precedence constraints	$l_{j,i}$		binary incidence matrix indicating job precedence constraints (j precedes i). The associated graph must be acyclic.
predecessors	$P(j) \subset J$		predecessors of job j , defined as the set of jobs that immediately precede job j (i. e., where $l_{j,i} = 1$)
successors	$S(j) \subset J$		successors of job j , defined as the set of jobs that immediately follow job j (i. e., where $l_{j,i} = 1$). We assume there is at most one successor per job.
device capacity	$K_{m,t}$	bytes/sec	total resource capacity available at device m
resource requirements	$R_{j,m}$	bytes/sec	resource requirements for job j on device m
job allocation duration	D_j	sec	duration of job j
<i>Decision variables</i>			
chosen path	$z_{w,p}$		binary indicator variable showing whether path p is chosen for workload w
chosen job	y_j		binary indicator variable showing whether job j is chosen
start time indicator	$s_{j,t}$		binary indicator describing whether job j starts in time slot t
end time indicator	$e_{j,t}$		binary indicator describing whether job j ends in time slot t (i. e., first slot where resources are released)
resource indicator	$r_{j,t}$		binary indicator describing whether job j consumes resources in time slot t

Table 4: Summary of notation used in discrete time MIP formulation for data recovery scheduling.

Device type	Bandwidth	Capacity
Disk array	512 MB/s	18.25 TB
Tape library	drives * 60 MB/s	tapes * 400 GB
OC3 network links	20 MB/s	n/a

Table 5: Device configuration parameters. The array characteristics are based on HP's EVA array [12], with 256 73 GB disks. The tape library (based on HP's ESL9595 [13]) contains up to 16 LTO tape drives and up to 500 LTO tape cartridges. These devices communicate through a fibre-channel storage area network (SAN). Sites are connected via OC3 links (155 Mbps). We assume that device reprovisioning takes 12 hours.

Workload	Resource requirements	
Application	local server	100%
	local array	1028 KB/s
Synchronous mirroring	local array	0 KB/s
	interconnect	7990 KB/s
	remote array	7990 KB/s
Intra-array split mirror	local array	3500 KB/s
Tape backup	local array	16,505 KB/s
	(local) tape library	16,505 KB/s

Table 6: Normal mode bandwidth resource demands. These bandwidths represent the minimal requirements for maintaining the schedules for creating secondary copies. Split mirrors are created every 12 hours, and four such split mirrors are retained at the workload's primary array. Each workload is backed up using weekly full backups (once a week, with a backup window of 24 hours) and daily incremental backups (once a day, with a backup window of 8 hours). If remote vaulting of backup tapes is used, we assume backup tapes are shipped offsite immediately. We assume that one server is allocated to each application workload.