The HP AutoRAID hierarchical storage system

John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan
Hewlett-Packard Laboratories

Configuring redundant disk arrays is a black art. To configure an array properly, a system administrator must understand the details of both the array and the workload it will support. Incorrect understanding of either, or changes in the workload over time, can lead to poor performance.

We present a solution to this problem: a two-level storage hierarchy implemented inside a single disk-array controller. In the upper level of this hierarchy, two copies of active data are stored to provide full redundancy and excellent performance. In the lower level, RAID 5 parity protection is used to provide excellent storage cost for inactive data, at somewhat lower performance.

The technology we describe in this paper, known as HP AutoRAID, automatically and transparently manages migration of data blocks between these two levels as access patterns change. The result is a fully redundant storage system that is extremely easy to use, is suitable for a wide variety of workloads, is largely insensitive to dynamic workload changes, and performs much better than disk arrays with comparable numbers of spindles and much larger amounts of front-end RAM cache. Because the implementation of the HP AutoRAID technology is almost entirely in software, the additional hardware cost for these benefits is very small.

We describe the HP AutoRAID technology in detail, provide performance data for an embodiment of it in a storage array, and summarize the results of simulation studies used to choose algorithms implemented in the array.

Categories and Subject Descriptors: B.4.2 [**Input/Output and Data Communications**]: Input/Output devices—*channels and controllers*; B.4.5 [**Input/Output and Data Communications**]: Reliability, Testing, and Fault-Tolerance—*redundant design*; D.4.2 [**Operating Systems**]: Storage Management—*secondary storage*

General Terms: Algorithms, Design, Performance, Reliability

Additional Key Words and Phrases: Disk array, RAID, storage hierarchy

1. INTRODUCTION

Modern businesses and an increasing number of individuals depend on the information stored in the computer systems they use. Even though modern disk drives have mean-time-to-failure (MTTF) values measured in hundreds of

Author's addresses: Hewlett-Packard Laboratories, mailstop 1U13, 1501 Page Mill Road, Palo Alto, CA 94304-1126; email: {wilkes,golding,staelin,sullivan}@hpl.hp.com.

a. RAID 3 **b**. RAID 5

Fig. 1. Data and parity layout for two different RAID levels.

years, storage needs have increased at an enormous rate, and a sufficiently large collection of such devices can still experience inconveniently frequent failures. Worse, completely reloading a large storage system from backup tapes can take hours or even days, resulting in very costly downtime.

For small numbers of disks, the preferred method to provide fault protection is to duplicate (*mirror*) data on two disks with independent failure modes. This solution is simple, and it performs well.

However, once the total number of disks gets large, it becomes more cost-effective to employ an array controller that uses some form of partial redundancy (such as parity) to protect the data it stores. Such RAIDs (for Redundant Arrays of Independent Disks) were first described in the early 1980s [Lawlor 1981; Park and Balasubramanian 1986], and popularized by the work of a group at UC Berkeley [Patterson et al. 1988; Patterson et al. 1989]. By storing only partial redundancy for the data, the incremental cost of the desired high availability is reduced to as little as 1/N of the total storage-capacity cost (where N is the number of disks in the array), plus the cost of the array controller itself.

The UC Berkeley RAID terminology has a number of different *RAID levels*, each one representing a different amount of redundancy and a placement rule for the redundant data. Most disk array products implement RAID level 3 or 5. In RAID level 3, host data blocks are bit- or byte-interleaved across a set of data disks, and parity is stored on a dedicated data disk (see Figure 1a). In RAID level 5, host data blocks are block-interleaved across the disks, and the disk on which the parity block is stored rotates in round-robin fashion for different stripes (see Figure 1b). Both hardware and software RAID products are available from many vendors.

Unfortunately, current disk arrays are often difficult to use [Chen and Lee 1993]: the different RAID levels have different performance characteristics and perform well only for a relatively narrow range of workloads. To accommodate this, RAID systems typically offer a great many configuration parameters: data- and parity-layout choice, stripe depth, stripe width, cache sizes and write-back policies, and so on. Setting these correctly is difficult: it requires knowledge of workload characteristics that most people are unable (and unwilling) to acquire. As a result, setting up a RAID array is often a daunting task that requires skilled, expensive people and—in too many cases—a painful process of trial and error.

Making the wrong choice has two costs: the resulting system may perform poorly; and changing from one layout to another almost inevitably requires copying data off to a second device, reformatting the array, and then reloading it. Each step of this process can take hours; it is also an opportunity for inadvertent data loss through operator error—one of the commonest source of problems in modern computer systems [Gray 1990].

Adding capacity to an existing array is essentially the same problem: taking full advantage of a new disk usually requires a reformat and data reload.

Since RAID 5 arrays suffer reduced performance in “degraded mode”—when one of the drives has failed—many include a provision for one or more spare disks that can be pressed into service as soon as an active disk fails. This allows redundancy reconstruction to commence immediately, thereby reducing the window of vulnerability to data loss from a second device failure and also minimizing the duration of the performance degradation. In the normal case, however, these spare disks are not used, and contribute nothing to the performance of the system. (There is also the secondary problem of assuming that a spare disk is still working: because the spare is idle, the array controller may not find out that it has failed until it is too late.)

1.1 The solution: a managed storage hierarchy

Fortunately, there is a solution to these problems for a great many applications of disk arrays: a redundancy-level storage hierarchy. The basic idea is to combine the performance advantages of mirroring with the cost-capacity benefits of RAID 5 by mirroring active data and storing relatively inactive or read-only data in RAID 5.

To make this solution work, part of the data must be active and part inactive (else the cost-performance would reduce to that of mirrored data), and the active subset must change relatively slowly over time (to allow the array to do useful work, rather than just move data between the two levels). Fortunately, studies on I/O access patterns, disk shuffling, and file-system restructuring have shown that these conditions are often met in practice [Akyurek and Salem 1993; Deshpandee and Bunt 1988; Floyd and Schlatter Ellis 1989; Geist et al. 1994; Majumdar 1984; McDonald and Bunt 1989; McNutt 1994; Ruemmler and Wilkes 1991; Ruemmler and Wilkes 1993; Smith 1981].

Such a storage hierarchy could be implemented in a number of different ways:

- *Manually*, by the system administrator. (This is how large mainframes have been run for decades. [Gelb 1989] discusses a slightly refined version of this basic idea.) The advantage of this approach is that human intelligence can be brought to bear on the problem, and perhaps knowledge that is not available to the lower levels of the I/O and operating systems. However, it is obviously error-prone (the wrong choices can be made, and mistakes can be made in moving data from one level to another); it cannot adapt to rapidly changing access patterns; it requires highly skilled people; and it does not allow new resources (such as disk drives) to be added to the system easily.

- *In the file system*, perhaps on a per-file basis. This might well be the best possible place in terms of a good balance of knowledge (the file system can track access patterns on a per-file basis) and implementation freedom. Unfortunately, there are many different file system implementations in customers' hands, so deployment is a major problem.
- *In a smart array controller*, behind a block-level device interface such as the Small Systems Computer Interface (SCSI) standard [SCSI 1991]. Although this level has the disadvantage that knowledge about files has been lost, it has the enormous compensating advantage of being easily deployable—strict adherence to the standard means that an array using this approach can look just like a regular disk array, or even just a set of plain disk drives.

Not surprisingly, we are describing an array-controller-based solution here. We use the name "HP AutoRAID" to refer both to the collection of technology developed to make this possible and to its embodiment in an array controller.

1.2 Summary of the features of HP AutoRAID

We can summarize the features of HP AutoRAID as follows:

Mapping. Host block addresses are internally mapped to their physical locations in a way that allows transparent migration of individual blocks.

Mirroring. Write-active data are mirrored for best performance and to provide single-disk failure redundancy.

RAID 5. Write-inactive data are stored in RAID 5 for best cost-capacity while retaining good read performance and single-disk failure redundancy. In addition, large sequential writes go directly to RAID 5 to take advantage of its high bandwidth for this access pattern.

Adaptation to changes in the amount of data stored. Initially, the array starts out empty. As data are added, internal space is allocated to mirrored storage until no more data can be stored this way. When this happens, some of the storage space is automatically reallocated to the RAID 5 storage class, and data are migrated down into it from the mirrored storage class. Since the RAID 5 layout is a more compact data representation, more data can now be stored in the array. This reapportionment is allowed to proceed until the capacity of the mirrored storage has shrunk to about 10% of the total usable space. (The exact number is a policy choice made by the implementors of the HP AutoRAID firmware to maintain good performance.) Space is apportioned in coarse-granularity 1MB units.

Adaptation to workload changes. As the active set of data changes, newly active data are promoted to mirrored storage, and data that have become less active are demoted to RAID 5 in order to keep the amount of mirrored data roughly constant. Because these data movements can usually be done in the background, they do not affect the performance of the array. Promotions and demotions occur completely automatically, in relatively fine-granularity 64KB units.

Hot-pluggable disks, fans, power supplies, and controllers. These allow a failed component to be removed and a new one inserted while the system continues to operate. Although these are relatively commonplace features in higher-end disk arrays, they are important in enabling the next three features.

On-line storage capacity expansion. A disk can be added to the array at any time, up to the maximum allowed by the physical packaging—currently 12 disks. The system automatically takes advantage of the additional space by allocating more mirrored storage. As time and the workload permit, the active data are rebalanced across the available drives to even out the workload between the newcomer and the previous disks—thereby getting maximum performance from the system.

Easy disk upgrades. Unlike conventional arrays, the disks do not all need to have the same capacity. This has two advantages: first, each new drive can be purchased at the optimal capacity/cost/performance point, without regard to prior selections. Second, the entire array can be upgraded to a new disk type (perhaps with twice the capacity) without interrupting its operation by removing one old disk at a time, inserting a replacement disk, and then waiting for the automatic data reconstruction and rebalancing to complete. To eliminate the reconstruction, data could first be "drained" from the disk being replaced: this would have the advantage of retaining continuous protection against disk failures during this process, but would require enough spare capacity in the system.

Controller fail-over. A single array can have two controllers, each capable of running the entire subsystem. On failure of the primary, the operations are rolled over to the other. A failed controller can be replaced while the system is active. Concurrently active controllers are also supported.

Active hot spare. The spare space needed to perform a reconstruction can be spread across all of the disks, and used to increase the amount of space for mirrored data—and thus the array's performance—rather than simply being left idle.

If a disk fails, mirrored data are demoted to RAID 5 to provide the space to reconstruct the desired redundancy. Once this process is complete, a second disk failure can be tolerated—and so on, until the physical capacity is entirely filled with data in the RAID 5 storage class.

Simple administration and setup. A system administrator can divide the storage space of the array into one or more logical units (LUNs in SCSI terminology) to correspond to the logical groupings of the data to be stored. Creating a new LUN or changing the size of an existing LUN is trivial: it takes about 10 seconds to go through the front-panel menus, select a size, and confirm the request. Since the array does not need to be formatted in the traditional sense, the creation of the LUN does not require a pass over all the newly allocated space to zero it and initialize its parity, an operation that can take hours in a regular array. Instead, all that is needed is for the controller's data structures to be updated.

Log-structured RAID 5 writes. A well-known problem of RAID 5 disk arrays is the so-called small-write problem. Doing an update-in-place of part of a stripe takes 4 I/Os: old data and parity have to be read, new parity calculated, and then new data and new parity written back. HP AutoRAID avoids this overhead in most cases by writing to its RAID 5 storage in a log-structured fashion—that is, only empty areas of disk are written to, so no old-data or old-parity reads are required.

1.3 Related work

Many papers have been published on RAID reliability, performance, and design variations for parity placement and recovery schemes (see [Chen et al. 1994] for an annotated bibliography). The HP AutoRAID work builds on many of these studies: we concentrate here on the architectural issues of using multiple RAID levels (specifically 1 and 5) in a single array controller.

Storage Technology Corporation's Iceberg [Ewing 1993; STK 1995] uses a similar indirection scheme to map logical IBM mainframe disks (count-key-data format) onto an array of 5.25" SCSI disk drives [Art Rudeseal, private communication, Nov. 1994]. Iceberg has to handle variable-sized records; HP AutoRAID has a SCSI interface and can handle the indirection using fixed-size blocks. The emphasis in the Iceberg project seems to have been on achieving extraordinarily high levels of availability; the emphasis in HP AutoRAID has been on performance once the single-component failure model of regular RAID arrays had been achieved. Iceberg does not include multiple RAID storage levels: it simply uses a single-level modified RAID 6 storage class [Dunphy et al. 1991; Ewing 1993].

A team at IBM Almaden has done extensive work in improving RAID array controller performance and reliability, and several of their ideas have seen application in IBM mainframe storage controllers. Their floating parity scheme [Menon and Kasson 1989; Menon and Kasson 1992] uses an indirection table to allow parity data to be written in a nearby slot, not necessarily its original location. This can help to reduce the small-write penalty of RAID 5 arrays. Their distributed sparing concept [Menon and Mattson 1992] spreads the spare space across all the disks in the array, allowing all the spindles to be used to hold data. HP AutoRAID goes further than either of these: it allows both data and parity to be relocated, and it uses the distributed spare capacity to increase the fraction of data held in mirrored form, thereby improving performance still further. Some of the schemes described in [Menon and Courtney 1993] are also used in the dual-controller version of the HP AutoRAID array to handle controller failures.

The Loge disk drive controller [English and Stepanov 1992], and its follow-ons Mime [Chao et al. 1992] and Logical Disk [de Jonge et al. 1993], all used a scheme of keeping an indirection table to fixed-sized blocks held on secondary storage. None of these supported multiple storage levels, and none was targeted at RAID arrays. Work on an Extended Function Controller at HP's disk divisions in the 1980s looked at several of these issues, but progress awaited development of suitable controller technologies to make the approach adopted in HP AutoRAID cost effective.

The log-structured writing scheme used in HP AutoRAID owes an intellectual debt to the body of work on log-structured file systems (LFS) [Carson and Setia 1992; Ousterhout and Douglis 1989; Rosenblum and Ousterhout 1992; Seltzer et al. 1993; Seltzer et al. 1995] and cleaning (garbage-collection) policies for them [Blackwell et al. 1995; McNutt 1994; Mogi and Kitsuregawa 1994].

There is a large literature on hierarchical storage systems and the many commercial products in this domain (for example [Chen 1973; Cohen et al. 1989; DEC 1993; Deshpandee and Bunt 1988; Epoch Systems Inc. 1988; Gelb 1989; Henderson and Poston 1989; Katz et al. 1991; Miller 1991; Misra 1981; Sienknecht et al. 1994; Smith 1981], together with much of the proceedings of the IEEE Symposia on Mass Storage Systems). Most of this work has been concerned with wider performance disparities between the levels than exist in HP AutoRAID. For example, such systems often use disk and robotic tertiary storage (tape or magneto-optical disk) as the two levels.

Several hierarchical storage systems have used front-end disks to act as a cache for data on tertiary storage. In HP AutoRAID, however, the mirrored storage is not a cache: instead data are moved between the storage classes, residing in precisely one class at a time. This method maximizes the overall storage capacity of a given number of disks.

The Highlight system [Kohl et al. 1993] extended LFS to two-level storage hierarchies (disk and tape) and also used fixed-size segments. Highlight's segments were around 1MB in size, however, and therefore were much better suited for tertiary-storage mappings than for two secondary-storage levels.

Schemes in which inactive data are compressed [Burrows et al. 1992; Cate 1990; Taunton 1991] exhibit some similarities to the storage-hierarchy component of HP AutoRAID but operate at the file system level rather than at the block-based device interface.

Finally, like most modern array controllers, HP AutoRAID takes advantage of the kind of optimizations noted in [Baker et al. 1991; Ruemmler and Wilkes 1993] that become possible with nonvolatile memory.

1.4 Roadmap to remainder of paper

The remainder of the paper is organized as follows. We begin with an overview of the technology: how an HP AutoRAID array controller works. Next come two sets of performance studies. The first is a set of measurements of a product prototype; the second a set of simulation studies used to evaluate algorithm choices for HP AutoRAID. Finally, we conclude the paper with a summary of the benefits of the technology.

2. THE TECHNOLOGY

This section of the paper introduces the basic technologies used in HP AutoRAID. It starts with an overview of the hardware, then discusses the layout of data on the disks of the array, including the structures used for mapping data blocks to their locations on disk. This is followed by brief descriptions of normal read and write operations to illustrate the flow of data through the system, and then by a series of operations that are (usually) performed in the background, to ensure that the performance of the system remains high over long periods of time.

2.1 The HP AutoRAID array controller hardware

An HP AutoRAID array is fundamentally similar to a regular RAID array. That is, it has a set of disks, an intelligent controller that incorporates a microprocessor, mechanisms for calculating parity, caches for staging data (some of which are nonvolatile), a connection to one or more host computers, and appropriate speed-matching buffers. Figure 2 is an overview of this hardware.

The hardware prototype for which we provide performance data uses four back-end SCSI buses to connect to its disks, and one or two fast-wide SCSI buses for its front-end host connection. Many other alternatives exist for packaging this technology, but are outside the scope of this paper.

The array presents one or more SCSI logical units (LUNs) to its hosts. Each of these is treated as a virtual device inside the array controller: their storage is freely intermingled. A LUN's size may be increased at any time (subject to capacity constraints). Not every block in a LUN must contain valid data—if nothing has been stored at an address, the array controller need not allocate any physical space to it.

2.2 Data layout

Much of the intelligence in an HP AutoRAID controller is devoted to managing data placement on the disks. A two-level allocation scheme is used.

2.2.1 *Physical data layout: PEGs, PEXes, and segments.* First, the data space on the disks is broken up into large-granularity objects called Physical EXtents (PEXes), as shown in Figure 3. PEXes are typically 1MB in size.

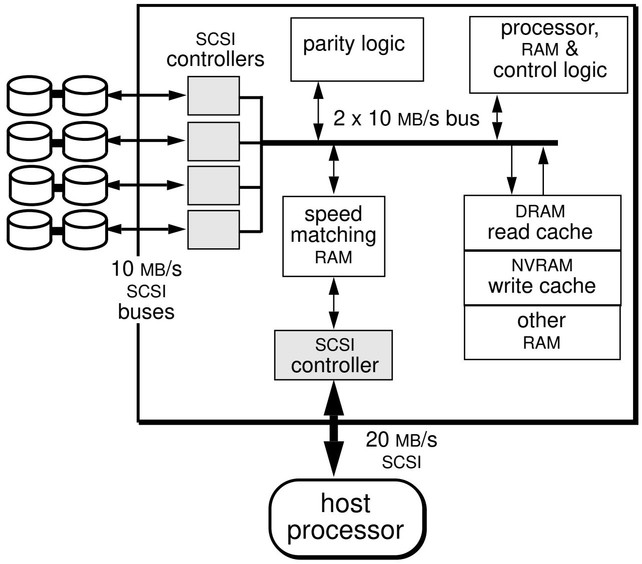


Fig. 2. Overview of HP AutoRAID hardware.

Several PEXes can be combined to make a Physical Extent Group (PEG). In order to provide enough redundancy to make it usable by either the mirrored or the RAID 5 storage class, a PEG includes at least three PEXes on different disks. At any given time, a PEG may be assigned to the mirrored storage class or the RAID 5 storage class, or may be unassigned, so we speak of mirrored, RAID 5, and free PEGs. (Our terminology is summarized in Table 1.)

Table 1: a summary of HP AutoRAID data-layout terminology.

Term	*Meaning*	*Size*
PEX (physical extent)	unit of physical space allocation	1MB
PEG (physical extent group)	a group of PEXes, assigned to one storage class	depends on number of disks
stripe	one row of parity and data segments in a RAID 5 storage class	depends on number of disks
segment	stripe unit (RAID 5) or half of a mirroring unit	128KB
RB (relocation block)	unit of data migration	64KB
LUN (logical unit)	host-visible virtual disk	user-settable

PEXes are allocated to PEGs in a manner that balances the amount of data on the disks (and thereby, hopefully, the load on the disks) while retaining the redundancy guarantees (no two PEXes from one disk can be used in the same stripe, for example). Because the disks in an HP AutoRAID array can be of different sizes, this allocation process may leave uneven amounts of free space on different disks.

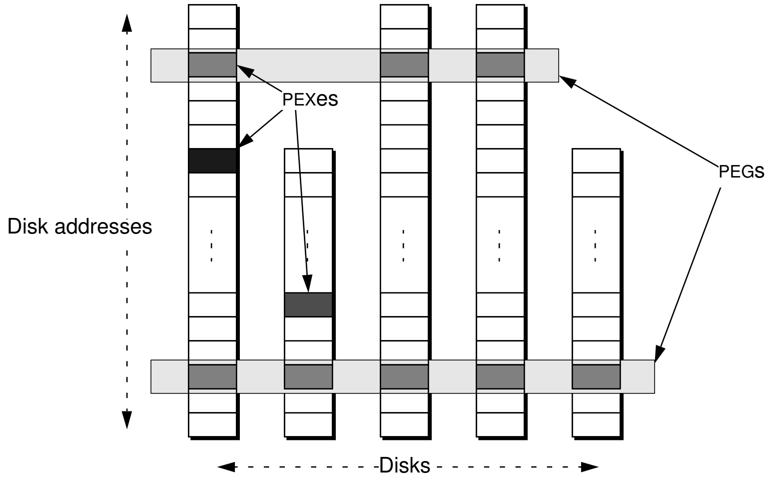


Fig. 3. Mapping of PEGs and PEXes onto disks (adapted from [Burkes et al. 1995]).

Segments are the units of contiguous space on a disk that are included in a stripe or mirrored pair; each PEX is divided into a set of 128KB segments. As Figure 4 shows, mirrored and RAID 5 PEGs are divided into segments in exactly the same way, but the segments are logically grouped and used by the storage classes in different ways: in RAID 5, a segment is the stripe unit; in the mirrored storage class, a segment is the unit of duplication.

2.2.2 *Logical data layout: RBs.* The logical space provided by the array—that visible to its clients—is divided into relatively small 64KB units called *Relocation Blocks* (RBs). These are the basic units of migration in the system. When a LUN is created or is increased in size, its address space is mapped onto a set of RBs. An RB is not assigned space in a particular PEG until the host issues a write to a LUN address that maps to the RB.

The size of an RB is a compromise between data layout, data migration, and data access costs. Smaller RBS require more mapping information to record where they have been put, and also increase the fraction of logically sequential accesses that is devoted to disk seek and rotational delays. Larger RBs will increase migration costs if only small amounts of data are updated in each RB. We report on our exploration of the relationship between RB size and performance in section 4.1.2.

Each PEG can hold many RBs, the exact number being a function of the PEG's size and its storage class. Currently unused RB slots in a PEG are marked free until they have an RB (i.e., data) allocated to them.

2.2.3 *Mapping structures.* A subset of the overall mapping structures are shown in Figure 5. These data structures are optimized for looking up the physical disk address of an RB, given its logical (LUN-relative) address, since that is the most common operation. In addition, data are held about access

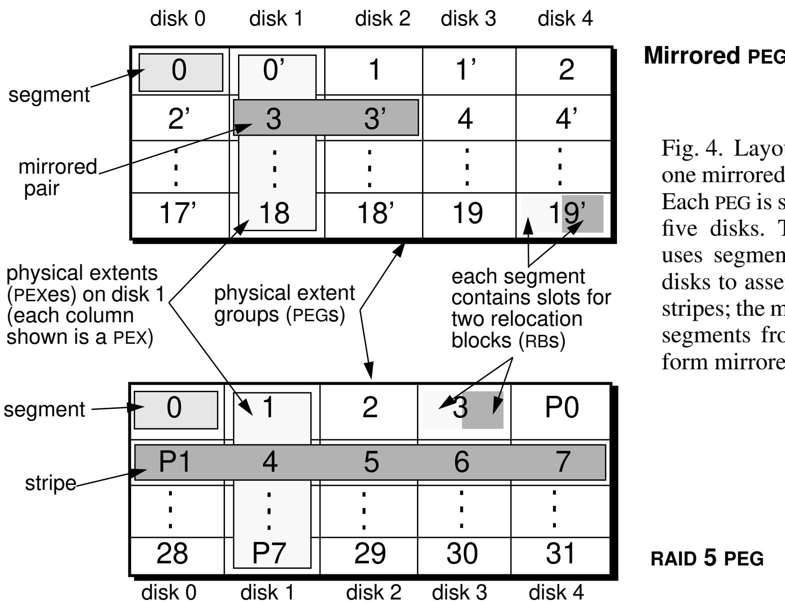


Fig. 4. Layout of two PEGs: one mirrored and one RAID 5. Each PEG is spread out across five disks. The RAID 5 PEG uses segments from all five disks to assemble each of its stripes; the mirrored PEG uses segments from two disks to form mirrored pairs.

times and history, the amount of free space in each PEG (for cleaning and garbage-collection purposes), and various other statistics. Not shown are various back pointers that allow additional scans.

2.3 Normal operations

To start a host-initiated read or write operation, the host sends a SCSI Command Descriptor Block (CDB) to the HP AutoRAID array, where it is parsed by the controller. Up to 32 CDBs may be active at a time. An additional 2,048 CDBs may be held in a FIFO queue waiting to be serviced; above this limit, requests are queued in the host. Long requests are broken up into 64KB pieces, which are handled sequentially; this method limits the amount of controller resources a single I/O can consume at minimal performance cost.

If the request is a read and the data are completely in the controller's cache memories, the data are transferred to the host via the speed-matching buffer, and the command then completes once various statistics have been updated. Otherwise, space is allocated in the front-end buffer cache, and one or more read requests are dispatched to the back-end storage classes.

Writes are handled slightly differently, because the nonvolatile front-end write buffer (*NVRAM*) allows the host to consider the request complete as soon as a copy of the data has been made in this memory. First a check is made to see if any cached data need invalidating, and then space is allocated in the NVRAM. This allocation may have to wait until space is available; in doing so,

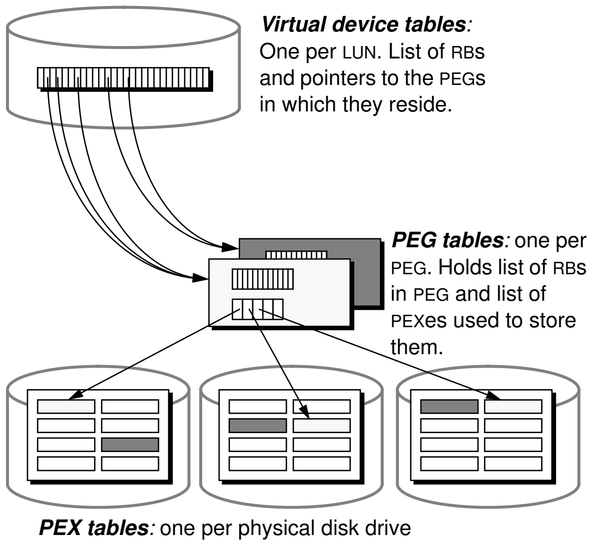


Fig. 5. Structure of the tables that map from addresses in virtual volumes to PEGs, PEXes, and physical disk addresses (simplified).

it will usually trigger a flush of existing dirty data to a back-end storage class. The data are transferred into the NVRAM from the host, and the host is then told that the request is complete. Depending on the NVRAM cache-flushing policy, a back-end write may be initiated at this point. More often, nothing is done, in the hope that another subsequent write can be coalesced with this one to increase efficiency.

Flushing data to a back-end storage class simply causes a back-end write of the data if they are already in the mirrored storage class. Otherwise, the flush will usually trigger a promotion of the RB from RAID 5 to mirrored. (There are a few exceptions that we describe later.) This promotion is done by calling the migration code, which allocates space in the mirrored storage class and copies the RB from RAID 5. If there is no space in the mirrored storage class (because the background daemons have not had a chance to run, for example), this may in turn provoke a demotion of some mirrored data down to RAID 5. There are some tricky details involved in ensuring that this cannot in turn fail—in brief, the free-space management policies must anticipate the worst-case sequence of such events that can arise in practice.

2.3.1 *Mirrored reads and writes.* Reads and writes to the mirrored storage class are straightforward: a read call picks one of the copies and issues a request to the associated disk. A write call causes writes to two disks; it returns only when both copies have been updated. Note that this is a back-end write call that is issued to flush data from the NVRAM, and is not synchronous with the host write.

2.3.2 *RAID 5 reads and writes.* Back-end reads to the RAID 5 storage class are as simple as for the mirrored storage class: in the normal case, a read is issued to the disk that holds the data. In the recovery case, the data may have to be reconstructed from the other blocks in the same stripe. (The usual RAID 5 recovery algorithms are followed in this case, so we will not discuss the failure case more in this paper. Although they are not implemented in the current system, techniques such as parity declustering [Holland and Gibson, 1992] could be used to improve recovery-mode performance.)

Back-end RAID 5 writes are rather more complicated, however. RAID 5 storage is laid out as a log: that is, freshly demoted RBs are appended to the end of a "current RAID 5 write PEG," overwriting virgin storage there. Such writes can be done in one of two ways: per-RB writes or batched writes. The former are simpler, the latter more efficient.

- For *per-RB writes*, as soon as an RB is ready to be written, it is flushed to disk. Doing so causes a copy of its contents to flow past the parity-calculation logic, which XORs it with its previous contents—the parity for this stripe. Once the data have been written, the parity can also be written. The prior contents of the parity block are stored in non-volatile memory during this process to protect against power failure. With this scheme, each data-RB write causes two disk writes: one for the data, one for the parity RB. This scheme has the advantage of simplicity, at the cost of slightly worse performance.
- For *batched writes*, the parity is written only after all the data RBs in a stripe have been written, or at the end of a batch. If, at the beginning of a batched write, there are already valid data in the PEG being written, the prior contents of the parity block are copied to nonvolatile memory along with the index of the highest-numbered RB in the PEG that contains valid data. (The parity was calculated by XORing only RBs with indices less than or equal to this value.) RBs are then written to the data portion of the stripe until the end of the stripe is reached or the batch completes; at that point, the parity is written. The new parity is computed on the fly by the parity calculation logic as each data RB is being written. If the batched write fails to complete for any reason, the system is returned to its pre-batch state by restoring the old parity and RB index, and the write is retried using the per-RB method.

  Batched writes require a bit more coordination than per-RB writes, but require only one additional parity write for each full stripe of data that is written. Most RAID 5 writes are batched writes.

In addition to these logging write methods, the method typically used in non-logging RAID 5 implementations (*read-modify-write*) is also used in some cases. This method, which reads old data and parity, modifies them, and rewrites them to disk, is used to allow forward progress in rare cases when no PEG is available for use by the logging write processes. It is also used when it is better to update data (or *holes*; see section 2.4.1) in place in RAID 5 than to migrate an RB into mirrored storage, such as in background migrations when the array is idle.

2.4 Background operations

In addition to the foreground activities described above, the HP AutoRAID array controller executes many background activities such as garbage collection and layout balancing. These background algorithms attempt to provide "slack" in the resources needed by foreground operations so that the foreground never has to trigger a synchronous version of these background tasks, since these can dramatically reduce performance.

The background operations are triggered when the array has been "idle" for a period of time. "Idleness" is defined by an algorithm that looks at current and past device activity—the array does not have to be completely devoid of activity. When an idle period is detected, the array performs one set of background operations. Each subsequent idle period, or continuation of the current one, triggers another set of operations.

After a long period of array activity, the current algorithm may need a moderate amount of time to detect that the array is idle. We hope to apply some of the results from [Golding et al. 1995] to improve idle-period detection and prediction accuracy, which will in turn allow us to be more aggressive about executing background algorithms.

2.4.1 *Compaction: cleaning and hole-plugging.* The mirrored storage class acquires *holes*, or empty RB slots, when RBs are demoted to the RAID 5 storage class. (Since updates to mirrored RBs are written in place, they generate no holes.) These holes are added to a free list in the mirrored storage class and may subsequently be used to contain promoted or newly created RBs. If a new PEG is needed for the RAID 5 storage class, and no free PEXes are available, a mirrored PEG may be chosen for *cleaning*: all the data are migrated out to fill holes in other mirrored PEGs, after which the PEG can be reclaimed and reallocated to the RAID 5 storage class.

Similarly, the RAID 5 storage class acquires holes when RBs are promoted to the mirrored storage class, usually because the RBs have been updated. Because the normal RAID 5 write process uses logging, the holes cannot be reused directly; we call them *garbage*, and the array needs to perform a periodic *garbage collection* to eliminate them.

If the RAID 5 PEG containing the holes is almost full, the array performs *hole-plugging* garbage collection. RBs are copied from a PEG with a small number of RBs and used to fill in the holes of an almost-full PEG. This minimizes data movement if there is a spread of fullness across the PEGs, which is often the case.

If the PEG containing the holes is almost empty and there are no other holes to be plugged, the array does PEG *cleaning*: that is, it appends the remaining valid RBs to the current end of the RAID 5 write log and reclaims the complete PEG as a unit.

2.4.2 *Migration: moving RBs between levels.* A background migration policy is run to move RBs from mirrored storage to RAID 5. This is done primarily to provide enough empty RB slots in the mirrored storage class to handle a future write burst. As [Ruemmler and Wilkes 1993] showed, such bursts are quite common.

RBs are selected for migration by an approximate Least-Recently-Written algorithm. Migrations are performed in the background until the number of free RB slots in the mirrored storage class or free PEGs exceeds a high-water mark that is chosen to allow the system to handle a burst of incoming data. This threshold can be set to provide better burst-handling at the cost of slightly lower out-of-burst performance. The current AutoRAID firmware uses a fixed value, but the value could also be determined dynamically.

2.4.3 *Balancing: adjusting data layout across drives.* When new drives are added to an array, they contain no data and therefore do not contribute to the system's performance. *Balancing* is the process of migrating PEXes between disks to equalize the amount of data stored on each disk, and thereby also the request load imposed on each disk. Access histories could be used to balance the disk load more precisely, but this is not currently done. Balancing is a background activity, performed when the system has little else to do.

Another type of imbalance results when a new drive is added to an array: newly created RAID 5 PEGS will use all of the drives in the system to provide maximum performance, but previously created RAID 5 PEGS will continue to use only the original disks. This imbalance is corrected by another low-priority background process that copies the valid data from the old PEGS to new, full-width PEGS.

2.5 Workload logging

One of the uncertainties we faced while developing the HP AutoRAID design was the lack of a broad range of real system workloads at the disk I/O level that had been measured accurately enough for us to use in evaluating its performance.

To help remedy this in the future, the HP AutoRAID array incorporates an I/O workload logging tool. When the system is presented with a specially formatted disk, the tool records the start and stop times of every externally issued I/O request. Other events can also be recorded, if desired. The overhead of doing this is very small: the event logs are first buffered in the controller's RAM and then written out in large blocks. The result is a faithful record of everything the particular unit was asked to do, which can be used to drive simulation design studies of the kind we describe later in this paper.

2.6 Management tool

The HP AutoRAID controller maintains a set of internal statistics, such as cache utilization, I/O times, and disk utilizations. These statistics are relatively cheap to acquire and store, and yet can provide significant insight into the operation of the system.

The product team developed an off-line, inference-based management tool that uses these statistics to suggest possible configuration choices. For example, the tool is able to determine that for a particular period of high load, performance could have been improved by adding cache memory because the array controller was short of read cache. Such information allows administrators to maximize the array's performance in their environment.

3. HP AutoRAID PERFORMANCE RESULTS

A combination of prototyping and event-driven simulation was used in the development of HP AutoRAID. Most of the novel technology for HP AutoRAID is embedded in the algorithms and policies used to manage the storage hierarchy. As a result, hardware and firmware prototypes were developed concurrently with event-driven simulations that studied design choices for algorithms, policies, and parameters to those algorithms.

The primary development team was based at the product division that designed, built, and tested the prototype hardware and firmware. They were supported by a group at HP Laboratories that built a detailed simulator of the hardware and firmware and used it to model alternative algorithm and policy choices in some depth. This organization allowed the two teams to incorporate new technology into products in the least possible time while still fully investigating alternative design choices.

In this section we present measured results from a laboratory prototype of a disk array product that embodies the HP AutoRAID technology. In section 4 we present a set of comparative performance analyses of different algorithm and policy choices that were used to help guide the implementation of the real thing.

3.1 Experimental setup

The baseline HP AutoRAID configuration on which we report was a 12-disk system with one controller and 24MB of controller data cache. It was connected via two fast-wide differential SCSI adapters to an HP 9000/K400 system with one processor and 512MB of main memory running release 10.0 of the HP-UX operating system [Clegg et al. 1986]. All the drives used were 2.0GB 7,200RPM Seagate ST32550 Barracudas with immediate write reporting turned off.

To calibrate the HP AutoRAID results against external systems, we also took measurements on two other disk subsystems. These measurements were taken on the same host hardware, on the same days, with the same host configurations, number of disks, and type of disks:

- A Data General CLARiiON® Series 2000 Disk-Array Storage System Deskside Model 2300 with 64MB front-end cache. (We refer to this system as "RAID array.") This array was chosen because it is the recommended third-party RAID array solution for one of the primary customers of the HP AutoRAID product.

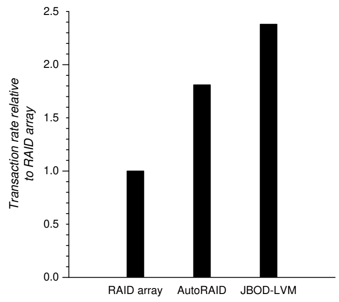
  Because the CLARiiON supports only one connection to its host, only one of the K400's fast-wide differential SCSI channels was used. The single channel was not, however, the bottleneck of the system.The array was configured to use RAID 5. (Results for RAID 3 were never better than for RAID 5.)
- A set of directly-connected individual disk drives. This solution provides no data redundancy at all. The HP-UX Logical Volume Manager (LVM) was used to stripe data across these disks in 4MB chunks. Unlike HP AutoRAID and the RAID array, the disks had no central controller and therefore no controller-level cache. We refer to this configuration as "JBOD-LVM" (Just a Bunch Of Disks).

3.2 Performance results

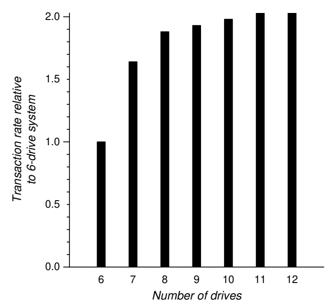
We begin by presenting some database macro-benchmarks in order to demonstrate that HP AutoRAID provides excellent performance for real-world workloads. Such workloads often exhibit behaviors such as burstiness that are not present in simple I/O rate tests; relying only on the latter can provide a misleading impression of how a system will behave in real use.

3.2.1 *Macro-benchmarks.* An OLTP database workload made up of medium-weight transactions was run against the HP AutoRAID array, the regular RAID array, and JBOD-LVM. The database used in this test was 6.7GB, which allowed it to fit entirely in mirrored storage in the HP AutoRAID; working set sizes larger than available mirrored space are discussed below. For this benchmark, the RAID array's 12 disks were spread evenly across its 5 SCSI channels, the 64MB cache was enabled, the cache page size was set to 2KB (the optimal value for this workload), and the default 64KB stripe-unit size was used. Figure 6a shows the result: HP AutoRAID significantly outperforms the RAID array and has performance about three-fourths that of JBOD-LVM. These results suggest that the HP AutoRAID is performing much as expected: keeping the data in mirrored storage means that writes are faster that the RAID array, but not as fast as JBOD-LVM. Presumably reads are being handled about equally well by all the cases.

Figure 6b shows HP AutoRAID's performance when data must be migrated between mirrored storage and RAID 5 because the working set is too large to be contained entirely in the mirrored storage class. The same type of OLTP database workload as described above was used, but the database size was set



(a) Comparison of HP AutoRAID and non-RAID drives with a regular RAID array. Both systems used 12 drives, and the entire 6.7GB database fit in RAID 1 in HP AutoRAID.



(b) Performance of HP AutoRAID when different numbers of drives are used. The fraction of the 8.1GB database held in mirrored storage was: 1/3 in the 6-drive system; 2/3 in the 7-drive system; nearly all in the 8-drive system; all in the larger systems.

Fig. 6. OLTP macro-benchmark results.

to 8.1GB. This would not fit in a 5-drive HP AutoRAID system, so we started with a 6-drive system as the baseline. Mirrored storage was able to accommodate one-third of the database in this case; two-thirds in the 7-drive system; almost all in the 8-drive system; and all of it in larger systems.

The differences in performance between the 6-, 7-, and 8-drive systems were due primarily to differences in the number of migrations performed, while the differences in the larger systems result from having more spindles across which to spread the same amount of mirrored data. The 12-drive configuration was limited by the host K400's CPU speed and performed about the same as the 11-drive system. From these data we see that even for this database workload, which has a fairly random access pattern across a large data set, HP AutoRAID performs within a factor of two of its optimum when only one-third of the data is held in mirrored storage, and at about three-fourths of its optimum when two-thirds of the data are mirrored.

3.2.2 *Micro-benchmarks.* In addition to the database macro-benchmark, we also ran some micro-benchmarks that used a synthetic workload-generation program known as DB to drive the arrays to saturation; the working-set size for the random tests was 2GB. These measurements were taken under slightly different conditions from the ones reported in section 3.1:

- The HP AutoRAID contained 16MB of controller data cache.
- An HP 9000/897 was the host for all the tests.

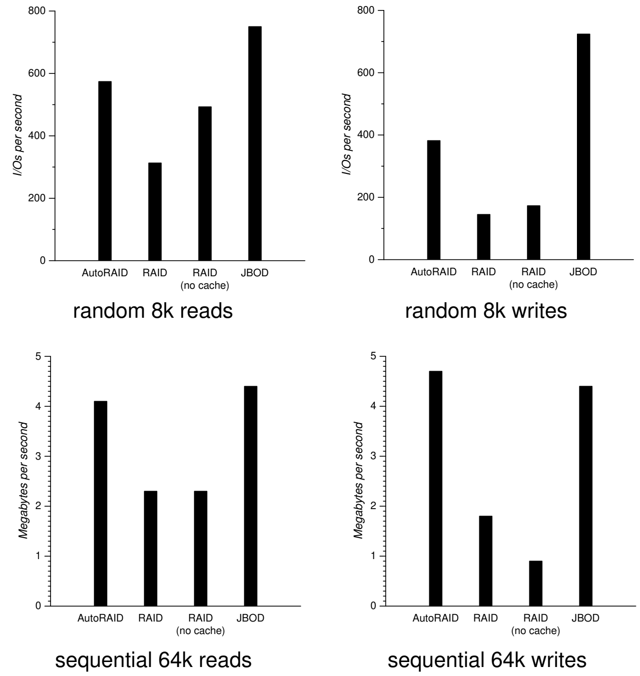


Fig. 7. Micro-benchmark comparisons of HP AutoRAID, a regular RAID array, and non-RAID drives.

- A single fast-wide differential SCSI channel was used for the HP AutoRAID and RAID array tests.
- The JBOD case did not use LVM, so did not do any striping. (Given the nature of the workload, this was probably immaterial.) In addition, 11 JBOD disks were used rather than 12 to match the amount of space available for data in the other configurations. Finally, the JBOD test used a fast-wide, single-ended SCSI card that required more host CPU cycles per I/O. We believe that this did not affect the micro-benchmarks because they were not CPU limited.
- The RAID array used 8KB cache pages, and cache on or off as noted.

Data from the micro benchmarks are provided in Figure 7. This shows the relative performance of the two arrays and JBOD for random and sequential reads and writes.

The random 8KB read-throughput test is primarily a measure of controller overheads. HP AutoRAID performance is roughly midway between the RAID array with its cache disabled and JBOD. It would seem that the cache searching algorithm of the RAID array is significantly limiting its performance, given that the cache hit rate would have been close to zero in these tests.

The random 8KB write-throughput test is primarily a test of the low-level storage system used since the systems are being driven into a disk-limited behavior by the benchmark. As expected, there is about a 1:2:4 ratio in I/Os per second for RAID 5 (4 I/Os for a small update): HP AutoRAID (2 I/Os to mirrored storage): JBOD (1 write in place).

The sequential 64KB read-bandwidth test shows that the use of mirrored storage in HP AutoRAID can largely compensate for controller overhead and deliver performance comparable to that of JBOD.

Finally, the sequential 64kb write-bandwidth test illustrates HP AutoRAID's ability to stream data to disk through its NVRAM cache: its performance is better than the pure JBOD solution.

We do not have a good explanation for the relatively poor performance of the RAID array in the last two cases; the results shown are the best obtained from a number of different array configurations. Indeed, the results demonstrated the difficulties involved in properly configuring a RAID array: many parameters were adjusted (caching on or off, cache granularity, stripe depth, and data layout), and no single combination performed well across the range of workloads examined.

3.2.3 *Thrashing.* As we noted in section 1.1, the performance of HP AutoRAID depends on the working-set size of the applied workload. With the working set within the size of the mirrored space, performance is very good, as shown by Figure 6a and Figure 7. And as Figure 6b shows, good performance can also be obtained when the entire working set does not fit in mirrored storage.

If the active write working-set exceeds the size of mirrored storage for long periods of time, however, it is possible to drive the HP AutoRAID array into a *thrashing* mode in which each update causes the target RB to be promoted up to the mirrored storage class, and a second one demoted to RAID 5. An HP AutoRAID array can usually be configured to avoid this by adding enough disks to keep all the write-active data in mirrored storage. If *all* the data were write-active, the cost-performance advantages of the technology would, of course, be reduced. Fortunately, it is fairly easy to predict or detect the environments that have a large write working-set and to avoid them if necessary. If thrashing does occur, HP AutoRAID detects it and reverts to a mode in which it writes directly to RAID 5—that is, it automatically adjusts its behavior so that performance is no worse than that of RAID 5.

4. SIMULATION STUDIES

In this section, we will illustrate several design choices that were made inside the HP AutoRAID implementation using a trace-driven simulation study.

Our simulator is built on the Pantheon [Cao et al. 1994; Golding et al. 1994] simulation framework,[1] which is a detailed, trace-driven simulation environment written in C++. Individual simulations are configured from the

[1] The simulator was formerly called TickerTAIP, but we have changed its name to avoid confusion with the parallel RAID array project of the same name [Cao et al. 1994].

set of available C++ simulation objects using scripts written in the Tcl language [Ousterhout 1994] and configuration techniques described in [Golding et al. 1994]. The disk models used in the simulation are improved versions of the detailed, calibrated models described in [Ruemmler and Wilkes 1994].

The traces used to drive the simulations are from a variety of systems, including: cello, a time-sharing HP 9000 Series 800 HP-UX system; snake, an HP 9000 Series 700 HP-UX cluster file server; OLTP, an HP 9000 Series 800 HP-UX system running a database benchmark made up of medium-weight transactions (not the system described in section 3.1); a personal workstation; and a Netware server. We also used subsets of these traces, such as the /usr disk from cello, a subset of the database disks from OLTP, and the OLTP log disk. Some of them were for long time periods (up to three months), although most of our simulation runs used two-day subsets of the traces. All but the Netware trace contained detailed timing information to 1μs resolution. Several of them are described in considerable detail in [Ruemmler and Wilkes 1993].

We modeled the hardware of HP AutoRAID using Pantheon components (caches, buses, disks, etc.) and wrote detailed models of the basic firmware and of several alternative algorithms or policies for each of about forty design experiments. The Pantheon simulation core comprises about 46k lines of C++ and 8k lines of Tcl, and the HP-AutoRAID-specific portions of the simulator added another 16k lines of C++ and 3k lines of Tcl.

Because of the complexity of the model and the number of parameters, algorithms, and policies that we were examining, it was impossible to explore all combinations of the experimental variables in a reasonable amount of time. We chose instead to organize our experiments into baseline runs and runs with one or a few related changes to the baseline. This allowed us to observe the performance effects of individual or closely related changes and to perform a wide range of experiments reasonably quickly. (We used a cluster of 12 high-performance workstations to run the simulations; even so, executing all of our experiments took about a week of elapsed time.)

We performed additional experiments to combine individual changes that we suspected might strongly interact (either positively or negatively) and to test the aggregate effect of a set of algorithms that we were proposing to the product development team.

No hardware implementation of HP AutoRAID was available early in the simulation study, so we were initially unable to calibrate our simulator (except for the disk models). Because of the high level of detail of the simulation, however, we were confident that *relative* performance differences predicted by the simulator would be valid even if *absolute* performance numbers were not yet calibrated. We therefore used the relative performance differences we observed in simulation experiments to suggest improvements to the team implementing the product firmware, and these are what we present

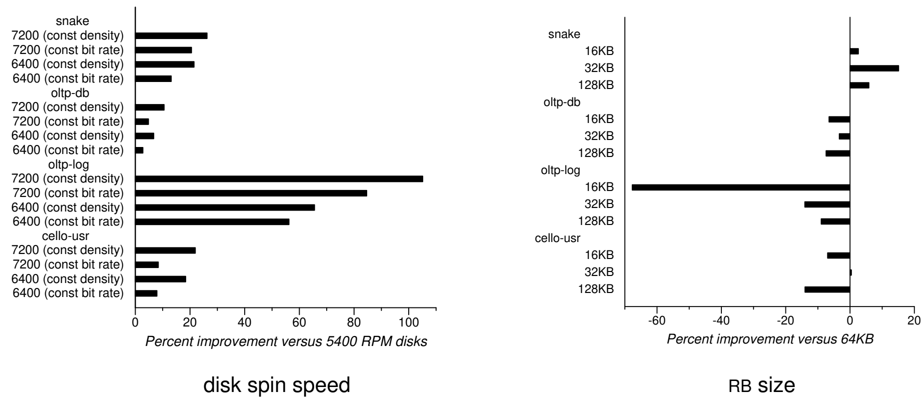


Fig. 8. Effects of disk spin speed and RB size on performance.

here. In turn, we updated our baseline model to correspond to the changes they made to their implementation.

Since there are far too many individual results to report here, we have chosen to describe a few that highlight some of the particular behaviors of the HP AutoRAID system.

4.1 Disk speed

Several experiments measured the sensitivity of the design to the size or performance of various components. For example, we wanted to understand whether faster disks would be cost-effective. The baseline disks held 2GB and spun at 5,400 RPM. We evaluated four variations of this disk: spinning at 6,400 RPM and 7,200 RPM, keeping either the data density (bits per inch) or transfer rate (bits per second) constant. As expected, increasing the back-end disk performance generally improves overall performance, as shown in Figure 8. The results suggest that improving transfer rate is more important than improving rotational latency.

4.2 RB size

The standard AutoRAID system uses 64KB RBs as the basic storage unit. We looked at the effect of using smaller and larger sizes. For most of the workloads (see Figure 8), the 64KB size was the best of the ones we tried: the balance between seek and rotational overheads versus data movement costs is about right. (This is perhaps not too surprising: the disks we are using have track sizes of around 64KB, and transfer sizes in this range will tend to get much of the benefit from fewer mechanical delays.)

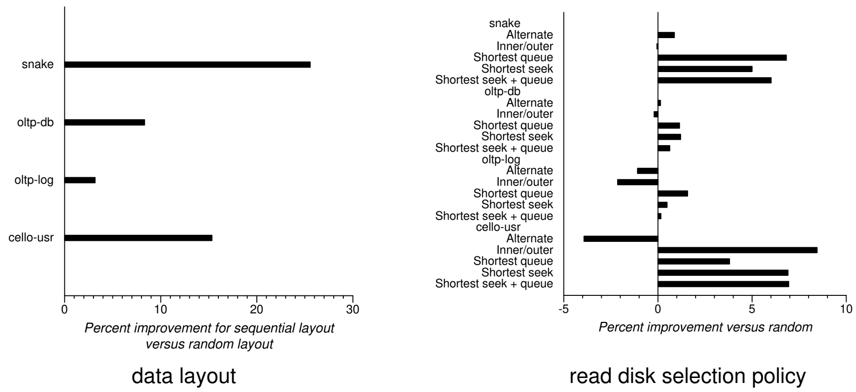


Fig. 9. Effects of data layout and mirrored storage class read disk selection policy on performance.

4.3 Data layout

Since the system allows blocks to be remapped, blocks that the host system has tried to lay out sequentially will often be physically discontiguous. To see how bad this problem could get, we compared the performance of the system when host LUN address spaces were initially laid out completely linearly on disk (as a best case) and completely randomly (as a worst case). Figure 9 shows the difference between the two layouts: there is a modest improvement in performance in the linear case compared with the random one. This suggests that the RB size is large enough to limit the impact of seek delays for sequential accesses.

4.4 Mirrored storage class read selection algorithm

When the front-end read cache misses on an RB that is stored in the mirrored storage class, the array can choose to read either of the stored copies. The baseline system selects the copy at random in an attempt to avoid making one disk a bottleneck. However, there are several other possibilities:

- strictly alternating between disks (alternate);
- attempting to keep the heads on some disks near the outer edge while keeping others near the inside (inner/outer);
- using the disk with the shortest queue (shortest queue);
- using the disk that can reach the block first, as determined by a shortest-positioning-time algorithm [Jacobson and Wilkes 1991; Seltzer et al. 1990] (shortest seek).

Further, the policies can be "stacked," using first the most aggressive policy but falling back to another to break a tie. In our experiments, random was always the final fallback policy.

Figure 9 shows the results of our investigations into the possibilities. By using shortest queue as a simple load-balancing heuristic, performance is improved

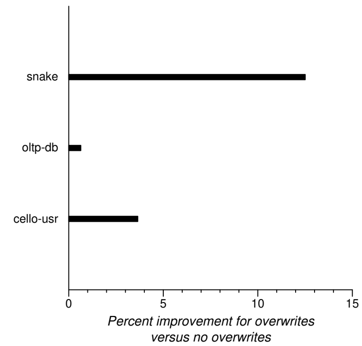


Fig. 10. Effect of allowing write cache overwrites on performance.

by an average of 3.3% over random for these workloads. Shortest seek performed 3.4% better than random on the average, but it is much more complex to implement because it requires detailed knowledge of disk head position and seek timing.

Static algorithms such as alternate and inner/outer sometimes perform better than random but sometimes interact unfavorably with patterns in the workload and decrease system performance.

We note in passing that these differences do not show up under micro-benchmarks (of the type reported in Figure 7) because the disks are typically always driven to saturation and do not allow such effects to show through.

4.5 Write cache overwrites

We investigated several policy choices for managing the NVRAM write cache. The baseline system, for instance, did not allow one write operation to overwrite dirty data already in cache; instead, the second write would block until the previous dirty data in the cache had been flushed to disk. As Figure 10 shows, allowing overwrites had a noticeable impact on most of the workloads. It had a huge impact on the OLTP-log workload, improving its performance by a factor of 5.3. We omitted this workload from the graph for scaling reasons.

4.6 Hole-plugging during RB demotion

RBs are typically written to RAID 5 for one of two reasons: demotion from mirrored storage, or garbage collection. During normal operation, the system creates holes in RAID 5 by promoting RBs to the mirrored storage class. In order to keep space consumption constant, the system later demotes (other) RBs to RAID 5. In the default configuration, HP AutoRAID uses logging writes to demote RBs to RAID 5 quickly, even if the demotion is done during idle time; these demotions do not fill the holes left by the promoted RBs. To reduce the work done by the RAID 5 cleaner, we allowed RBs demoted during idle periods to be written to RAID 5 using hole-plugging. This optimization reduced the number of RBs moved by the RAID 5 cleaner by 93% for the cello-usr workload and by 96% for snake, and improved mean I/O time for user I/Os by 8.4% and 3.2%.

5. SUMMARY

The HP AutoRAID technology works extremely well, providing performance close to that the of a nonredundant disk array across many workloads. At the same time, it provides full data redundancy and can tolerate failures of any single array component.

It is very easy to use: one of the authors of this paper was delivered a system without manuals a day before a demonstration, and had it running a trial benchmark five minutes after getting it connected to his completely unmodified workstation. The product team has had several such experiences in demonstrating the system to potential customers.

The HP AutoRAID technology is not a panacea for all storage problems: there are workloads that do not suit its algorithms well, and environments where the variability in response time is unacceptable. Nonetheless, it is able to adapt to a great many of the environments that are encountered in real life, and it provides an outstanding general-purpose storage solution where availability matters.

The first product based on the technology, the HP XLR1200 Advanced Disk Array, is now available.

Acknowledgments

We would like to thank our colleagues in HP's Storage Systems Division. They developed the HP AutoRAID system architecture and the product version of the controller, and were the customers for our performance and algorithm studies. Many more people put enormous amounts of effort into making this program a success than we can possibly acknowledge directly by name; we thank them all.

Chris Ruemmler wrote the DB benchmark used for the results in section 3.2.

This paper is dedicated to the memory of our late colleague Al Kondoff, who helped establish the collaboration that produced this body of work.

References

AKYÜREK, S., AND SALEM, K. 1993. Adaptive block rearrangement. Tech. Rep. CS–TR–2854.1, Department of Computer Science, Univ. of Maryland, College Park, Maryland.

BAKER, M., ASAMI, S., DEPRIT, E., OUSTERHOUT, J., AND SELTZER, M. 1992. Non-volatile memory for fast, reliable file systems. In *Proceedings of 5th International Conference on Architectural Support for Programming Languages and Operating Systems*. *Comput. Arch. News 20*, Oct., 10–22.

BLACKWELL, T., HARRIS, J., AND SELTZER, M. 1995. Heuristic cleaning algorithms in log-structured file systems. In *Proceedings of USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems*. USENIX Association, Berkeley, Calif., 277–88.

BURKES, T., DIAMOND, B., AND VOIGT, D. 1995. *Adaptive hierarchical RAID: a solution to the* RAID 5 *write problem*. Part No. 5963–9151. Hewlett-Packard Storage Systems Division, Boise, Idaho.

BURROWS, M., JERIAN, C., LAMPSON, B., AND MANN, T. 1992. On-line data compression in a log-structured file system. In *Proceedings of 5th International Conference on Architectural Support for Programming Languages and Operating Systems. Comput. Arch. News 20,* Oct., 2–9.

CAO, P., LIM, S. B., VENKATARAMAN, S., AND WILKES, J. 1994. The TickerTAIP parallel RAID architecture. In *ACM Trans. Comput. Syst. 12,* 3 (Aug.), 236–269.

CARSON, S., AND SETIA, S. 1992. Optimal write batch size in log-structured file systems. In *USENIX Workshop on File Systems.* USENIX Association, Berkeley, Calif., 79–91.

CATE, V. 1990. Two levels of filesystem hierarchy on one disk. Tech. Rep. CMU–CS–90–129. Carnegie-Mellon Univ., Pittsburgh, Penn.

CHAO, C., ENGLISH, R., JACOBSON, D., STEPANOV, A., AND WILKES, J. 1992. Mime: a high performance storage device with strong recovery guarantees. Tech. Rep. HPL–92–44. Hewlett-Packard Laboratories, Palo Alto, Calif.

CHEN, P. 1973. Optimal file allocation in multi-level storage hierarchies. In *Proceedings of National Computer Conference*. 277–82.

CHEN, P. M., AND LEE, E. K. 1993. Striping in a RAID level-5 disk array. Tech. Rep. CSE–TR–181–93. The Univ. of Michigan, Ann Arbor, Mich.

CHEN, P. M., LEE, E. K., GIBSON, G. A., KATZ, R. H., AND PATTERSON, D. A. 1994. RAID: high-performance, reliable secondary storage. In *ACM Comput. Surv. 26,* 2 (June), 145–85.

CLEGG, F. W., HO, G. S.-F., KUSMER, S. R., AND SONTAG, J. R. 1986. The HP-UX operating system on HP Precision Architecture computers. In *Hewlett-Packard Journal 37,* 12 (Dec.). Hewlett-Packard Company, Palo Alto, Calif., 4–22.

COHEN, E. I., KING, G. M., AND BRADY, J. T. 1989. Storage hierarchies. In *IBM Systems Journal 28,* 1. IBM Corp., Armonk, New York, 62–76.

DEC. 1993. *POLYCENTER storage management for OpenVMS VAX systems.* Digital Equipment Corp., Maynard, Mass.

DE JONGE, W., KAASHOEK, M. F., AND HSIEH, W. C. 1993. The Logical Disk: a new approach to improving file systems. In *Proceedings of 14th ACM Symposium on Operating Systems Principles*. ACM, New York, 15–28.

DESHPANDE, M. B., AND BUNT, R. B. 1988. Dynamic file management techniques. In *Proceedings of 7th IEEE Phoenix Conference on Computers and Communication.* IEEE, New York, 86–92.

DUNPHY, R. H. JR., WALSH, R., BOWERS, J. H., AND RUDESEAL, G. A. 1991. Disk drive memory. U. S. Patent 5,077,736; filed 13 Feb. 1990, granted 31 December 1991.

ENGLISH, R. M., AND STEPANOV, A.A. 1992. Loge: a self-organizing storage device. In *Proceedings of USENIX Winter'92 Technical Conference*. USENIX Association, Berkeley, Calif., 237–251.

EPOCH SYSTEMS INC. 1988. Mass storage: server puts optical discs on line for workstations. In *Electronics*, November.

EWING, J. 1993. *RAID: an overview.* Part No. W I7004-A 09/93. Storage Technology Corporation, Louisville, Colo. Available as http://www.stortek.com:80/StorageTek/raid.html.

FLOYD, R. A., AND SCHLATTER ELLIS, C. 1989. Directory reference patterns in hierarchical file systems. In *IEEE Trans. Know. Data Eng. 1,* 2 (June), 238–247.

GEIST, R., REYNOLDS, R., AND SUGGS, D. 1994. Minimizing mean seek distance in mirrored disk systems by cylinder remapping. In *Performance Evaluation 20,* 1–3 (May), 97–114.

GELB, J. P. 1989. System managed storage. In *IBM Systems Journal 28,* 1. IBM Corp., Armonk, New York, 77–103.

GOLDING, R., STAELIN, C., SULLIVAN, T., AND WILKES, J. 1994. "Tcl cures 98.3% of all known simulation configuration problems" claims astonished researcher! In *Proceedings of Tcl/Tk Workshop*. Available as Tech. Rep. HPL–CCD–94–11, Concurrent Computing Department, Hewlett-Packard Laboratories, Palo Alto, Calif.

GOLDING, R., BOSCH, P., STAELIN, C., SULLIVAN, T., AND WILKES, J. 1995. Idleness is not sloth. In *Proceedings of USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems*. USENIX Association, Berkeley, Calif., 201–212.

GRAY, J. 1990. A census of Tandem system availability between 1985 and 1990. Tech. Rep. 90.1. Tandem Computers Incorporated, Cupertino, Calif.

HENDERSON, R. L., AND POSTON, A. 1989. MSS-II and RASH: a mainframe Unix based mass storage system with a rapid access storage hierarchy file management system. In *Proceedings of USENIX Winter 1989 Conference*. USENIX Association, Berkeley, Calif., 65–84.

HOLLAND, M., AND GIBSON, G.A. 1992. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of 5th International Conference on Architectural Support for Programming Languages and Operating Systems*. *Comput. Arch. News 20*, Oct., 23–35.

JACOBSON, D. M., AND WILKES, J. 1991. Disk scheduling algorithms based on rotational position. Tech. Rep. HPL–CSP–91–7. Hewlett-Packard Laboratories, Palo Alto, Calif.

KATZ, R. H., ANDERSON, T. E., OUSTERHOUT, J. K., AND PATTERSON, D. A. 1991. Robo-line storage: low-latency, high capacity storage systems over geographically distributed networks. UCB/CSD 91/651. Computer Science Div., Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, Calif.

KOHL, J. T., STAELIN, C., AND STONEBRAKER, M. 1993. HighLight: using a log-structured file system for tertiary storage management. In *Proceedings of Winter 1993 USENIX*. USENIX Association, Berkeley, Calif., 435–447.

LAWLOR, F. D. 1981. Efficient mass storage parity recovery mechanism. In *IBM Technical Discl. Bulletin 24*, 2 (July). IBM Corp., Armonk, New York, 986–987.

MAJUMDAR, S. 1984. *Locality and file referencing behaviour: principles and applications*. MSc thesis published as Tech. Rep. 84–14. Department of Computer Science, University of Saskatchewan, Saskatoon, Saskatchewan.

MCDONALD, M. S., AND BUNT, R. B. 1989. Improving file system performance by dynamically restructuring disk space. In *Proceedings of Phoenix Conference on Computers and Communication.* IEEE, New York, 264–269.

MCNUTT, B. 1994. Background data movement in a log-structured disk subsystem. *IBM J. Res. and Development 38,* 1. IBM Corp., Armonk, New York, 47–58.

MENON, J., AND KASSON, J. 1989. Methods for improved update performance of disk arrays. Tech. Rep. RJ 6928 (66034). IBM Almaden Research Center, San Jose, Calif. Declassified 21 Nov. 1990.

MENON, J., AND KASSON, J. 1992. Methods for improved update performance of disk arrays. In *Proceedings of 25th International Conference on System Sciences*. Vol. 1. IEEE, New York, 74–83.

MENON, J., AND MATTSON, D. 1992. Comparison of sparing alternatives for disk arrays. In *Proceedings of 19th International Symposium on Computer Architecture*. ACM, New York, 318–329.

MENON, J., AND COURTNEY, J. 1993. The architecture of a fault-tolerant cached RAID controller. In *Proceedings of 20th International Symposium on Computer Architecture*. ACM, New York, 76–86.

MILLER, E. L. 1991. File migration on the Cray Y-MP at the National Center for Atmospheric Research. UCB/CSD 91/638, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, Calif.

MISRA, P. N. 1981. Capacity analysis of the mass storage system. In *IBM Systems Journal 20,* 3. IBM Corp., Armonk, New York, 346–361.

MOGI, K., AND KITSUREGAWA, M. 1994. Dynamic parity stripe reorganizations for RAID5 disk arrays. In *Proceedings of Parallel and Distributed Information Systems International Conference.* IEEE, New York, 17–26.

OUSTERHOUT, J., AND DOUGLIS, F. 1989. Beating the I/O bottleneck: a case for log-structured file systems. In *Operating Systems Review 23,* 1 (Jan.), 11–27.

OUSTERHOUT, J. K. 1994. *Tcl and the Tk toolkit*, Professional Computing series. Addison-Wesley, Reading, Mass. and London.

PARK, A., AND BALASUBRAMANIAN, K. 1986. Providing fault tolerance in parallel secondary storage systems. Tech. Rep. CS–TR–057–86. Department of Computer Science, Princeton University, Princeton, New Jersey.

PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. 1988. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of 1988 ACM SIGMOD International Conference on Management of Data.* ACM, New York.

PATTERSON, D. A., CHEN, P., GIBSON, G., AND KATZ, R. H. 1989. Introduction to redundant arrays of inexpensive disks (RAID). In *Spring COMPCON '89*. IEEE, New York, 112–117.

ROSENBLUM, M., AND OUSTERHOUT, J. K. 1992. The design and implementation of a log-structured file system. In *ACM Trans. on Comput. Syst. 10,* 1 (Feb.), 26–52.

RUEMMLER, C., AND WILKES, J. 1991. Disk shuffling. Tech. Rep. HPL–91–156, Hewlett-Packard Laboratories, Palo Alto, Calif.

RUEMMLER, C., AND WILKES, J. 1993. UNIX disk access patterns. In *Proceedings of Winter 1993 USENIX*. USENIX Association, Berkeley, Calif., 405–420.

RUEMMLER, C., AND WILKES, J. 1994. An introduction to disk drive modeling. In *IEEE Computer 27,* 3 (March), 17–28.

SCSI. 1991. Secretariat, Computer and Business Equipment Manufacturers Association 1991. *Draft proposed American National Standard for information systems – Small Computer System Interface-2 (SCSI-2)*, Draft ANSI standard X3T9.2/86-109., 2 February 1991 (revision 10d).

SELTZER, M., CHEN, P., AND OUSTERHOUT, J. 1990. Disk scheduling revisited. In *Proceedings of Winter 1990 USENIX Conference.* USENIX Association, Berkeley, Calif., 313–323.

SELTZER, M., BOSTIC, K., MCKUSICK, M. K., AND STAELIN, C. 1993. An implementation of a log-structured file system for UNIX. In *Proceedings of Winter 1993 USENIX*. USENIX Association, Berkeley, Calif., 307–326.

SELTZER, M., SMITH, K. A., BALAKRISHNAN, H., CHANG, J., MCMAINS, S., AND PADMANABHAN, V. 1995. File system logging versus clustering: a performance comparison. In *Conference Proceedings of USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems*. USENIX Association, Berkeley, Calif., 249–264.

SIENKNECHT, T. F., FRIEDRICH, R. J., MARTINKA, J. J., AND FRIEDENBACH, P. M. 1994. The implications of distributed data in a commercial environment on the design of hierarchical storage management. In *Performance Evaluation 20,* 1–3 (May). North-Holland, Amsterdam, 3–25.

SMITH, A. J. 1981. Optimization of I/O systems by cache disks and file migration: a summary. In *Performance Evaluation 1.* North-Holland, Amsterdam, 249–262.

STK. 1995. *Iceberg 9200 disk array subsystem*. Storage Technology Corporation, Louisville, Colo. Available as http://www.stortek.com:80/StorageTek/iceberg.html.

TAUNTON, M. 1991. Compressed executables: an exercise in thinking small. In *Proceedings of Summer USENIX.* USENIX Association, Berkeley, Calif., 385–403.