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 article, 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; C.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 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 of providing 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 RAID systems (for Redundant Arrays of Independent Disks) were first described in the early 1980s (Lawler 1981; Park and Balasubramanian 1986) and popularized by the work of a group at UC Berkeley (Patterson et al. 1988, 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-interleaved across a set of data disks, and parity is stored on a dedicated data disk (see Figure 11a). 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 11b). 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: disk 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, reformattting 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 common sources 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 reduc-
ing the window of vulnerability to data loss from a second device failure and 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 [Akayrek and Salem 1993; Deshpande and Bunt 1988; Floyd and Schlatter 1989; Geliat et al. 1994; Najmendar 1984; McDonald and Bunt 1989; McMillan 1994; Riemer and Wilkes 1991; 1993; Smith 1991].

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, below a block-level device interface such as the Small System 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 more compact data representation, more data can now be stored in the array. This reassignment 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 implementers of the HP AutoRAID firmware to maintain good performance.) Space is apportioned in coarse-granularity LMB 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-Plugable Disks, Fans, Power Supplies, and Controllers.** These allow 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 newcomers 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 one 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.

**Application-Set Space.** 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 indirect-write scheme to map logical IBM mainframe disks (count-key-data format) onto an array of 5.25-inch SCSI disk drives (Art Rudee et al., 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 Mattson 1989, 1992) uses an indirect 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 Lode disk drive controller (English and Stepanov 1992) and its follow-ons Minu (Chao et al. 1992) and Logical Disk (de Jonge et al. 1993) all used a scheme of keeping an indirect table of 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 1990s 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 Douglass 1989; Rosenblum and Ousterhout 1993).

There is a large body of literature on hierarchical storage systems and the many commercial products in this domain (for example, Chen [1973], Cohen et al. [1989], DEC [1985], Deshpandee and Bunt [1988], Epoch Systems [1988], Gelb [1989], Henderson and Poston [1989], Katz et al. [1991], Miller [1991], Misra [1981], Sieweke et al. [1994], and Smith [1991], together with much of the Proceedings of the IEEE Symposium on Mass Storage Systems). Most of this work has been concerned with wider performance disparity in the levels than exist in HP AutoRAID. For example, such systems often use disk and robotic tertiary storage (tape or magneto-optical disc) 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 used fixed-size segments. Highlight's segments were around 1 MB 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; Taunting 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] and Ruzemeier and Wilkes [1993] that become possible with nonvolatile memory.

1.4 Roadmap to Remainder of Artice

The remainder of the article 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 is a set of simulation studies used to evaluate algorithm choices for HP AutoRAID. Finally, we conclude the article with a summary of the benefits of the technology.

2. THE TECHNOLOGY

This section 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 an overview of normal read and write operations to illustrate the flow of data through the system, and then by descriptions of 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.


Fig. 2. Overview of HP AutoRAID hardware

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 storing data (some of which are volatile), 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 article.

The array presents one or more SCSI logical units (LU) to its hosts. Each of these is treated as a virtual device inside the array controller: their storage is freely intermingled. A LU's size may be increased at any time (subject to capacity constraints). Not every block in a LU 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: PEWs, 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 1 MB in size.

Several PEXes can be combined to form a Physical Extent Group (PEG). In order to provide enough redundancy to make it usable by either the mirrored RAID 5 storage class 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.)

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.

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 94KB units called Relocation Blocks (RBs). These are the basic units of migration in the system. When a LUN is created or its size is increased, its address space is mapped onto a set of RBs. An RB is not assigned anywhere 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 increase the time spent on 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.

Table 1. A Summary of HP AutoRAID Data Layout Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEX (physical extent)</td>
<td>Unit of physical space allocation. A group of PEXes, assigned to one</td>
<td>1MB</td>
</tr>
<tr>
<td>PEG (physical extent group)</td>
<td>A group of PEXes, assigned to storage class.</td>
<td>*</td>
</tr>
<tr>
<td>Stripe</td>
<td>One row of parity and data segments in a RAID 5 storage class.</td>
<td>*</td>
</tr>
<tr>
<td>Segment</td>
<td>Stripe unit (RAID 5) or half of a mirroring unit.</td>
<td>128KB</td>
</tr>
<tr>
<td>RB (relocation block)</td>
<td>Unit of data migration. Host-visible virtual disk.</td>
<td>64KB</td>
</tr>
<tr>
<td>LUN (logical unit)</td>
<td></td>
<td>User settable</td>
</tr>
</tbody>
</table>

* Depends on the number of disks.
Each PEG can hold many RBs, the exact number being a function of the PEG's size and its storage class. 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 is 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 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 an 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 2048 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 (NV-RAM) 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 NV-RAM. This allocation may have to wait until space is available; in doing so, it will usually trigger a flush of existing dirty data to a back-end storage class. The data are transferred into the NV-RAM from the host, and the host is then told that the request is complete. Depending on the NV-RAM 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 NV-RAM 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 reconstituted from the other blocks in the same stripe. The usual RAID 5 recovery algorithms are followed in this case, as we will not discuss the failure case more in this article. 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 denoted 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 nonvolatile memory during this process to protect against power failure. With this scheme, each data-RAI write causes two disk writes: one for the data and one for the parity RAI. 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 RAs in a stripe have been written, or at the end of a batch. If, at the beginning of a batch 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 RAI in the PEG that contains valid data. (The parity was calculated by XORing only RAs with indices less than or equal to this value.) RAs are then written to the data portion of the stripe until the end of the stripe is reached or until 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 RAI is being written. If the batched write fails to complete for any reason, the system is returned to its prebatch state by restoring the old parity and RAI index, and the write is retried using the per-RAI method.

Batched writes require a bit more coordination than per-RAI 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 nonlogging 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 RAI 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, which 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 Godding 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 RAI slots, when RAs are demoted to the RAID 5 storage class. (Since updates to mirrored RAs 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 RAs. If a new PEG is needed for the RAID 5 storage class, and no free PEEXs 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 RAs are promoted to the mirrored storage class, usually because the RAs 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. RAs are copied from a PEG with a small number of RAs 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 RAs to the current end of the RAID 5 write log and reclaims the complete PEG as a unit.

2.4.2 Migration: Moving RAs Between Levels. A background migration policy is run to move RAs from mirrored storage to RAID 5. This is done primarily to provide enough empty RAI slots in the mirrored storage class to handle a future write burst. As Rasmussen and Wilkes [1995] showed, such bursts are quite common.

RAs are selected for migration by an approximate Least Recently Written algorithm. Migrations are performed in the background until the number of free RAI slots in the mirrored storage class or free PEEXs 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 the array, they contain no data and therefore do not contribute to the system's performance. Balancing is the process of migrating PEEXs 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 studies of the kind we describe later in this article.

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 offline, 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 AUTO RAID 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 2.9GB 7200RPM Seagate ST9250-3A Seagate 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 Desk-side 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 5.1.

A set of directly connected individual disk drives. This system 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 “JBO (JBO = Just A Batch Of Disks).”

3.2 Performance Results

We begin by presenting some database macrobenchmarks 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 Microbenchmarks. 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, (1) the RAID array's 12 disks were spread evenly across its 5 SCSI channels, (2) the 64MB cache was enabled, (3) the cache page size was set to 2KB (the optimal value for this workload), and (4) the default 64KB stripe-unit size was used. Figure 6(a) shows the result: HP AutoRAID significantly outperforms the RAID array and has performance about three-fourths of 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 than the Raid array, but not as fast as JBOD-LVM. Presumably, reads are being handled about equally well by all the cases.

Figure 6(b) 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 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 K4000'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 Microbenchmarks. In addition to the database microbenchmark, we ran some microbenchmarks 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 5.1:

- The HP AutoRAID contained 16MB of controller data cache.
- An HP 9000/987 was the host for all the tests.
- 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 it 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 single-ended SCSI card that required more host CPU cycles per I/O. We believe that this did not affect the microbenchmarks because they were not CPU limited.

The RAID array used 8KB cache pages and cache on or off as noted.

Data from the microbenchmarks 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 6 (4 I/Os for a small update). HP AutoRAID (2 I/Os to mirrored storage) JBOG (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 JBOG.

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 JBOG 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 6(a) and Figure 7. And as Figure 6(b) 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 a long period 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; Goldberg et al. 1994] simulation framework, which is a detailed, trace-driven simulation environment written in C++. Individual simulations are configured from the set of available C++ simulation objects using scripts written in the Tcl language [Ousterhout 1994] and configuration techniques described in Goldberg et al. [1994]. The disk models used in the simulation are improved versions of the detailed, calibrated models described in Buehler and Wilkes [1994].

The traces used to drive the simulations are from a variety of systems, including: c060, a time-sharing HP 9000 Series 800 HP-UX system; snake, an HP 9000 Series 700 HP-UX cluster file server; OLT, an HP 9000 Series 800 HP-UX system running a database benchmark made up of medium-weight transactions (test the system described in Section 3.1); bddoes, a personal workstaton; and a Noteware server. We also used subsets of these traces, such as the /usr disk from c060, a subset of the database disks from OLT, and the OLT HP-UX 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 1 μs resolution. Several of them are described in considerable detail by Rasmussen and Wilkes [1988].

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 40 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 included 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, we were unable 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, all our experiments took about a week of elapsed time.)

We performed additional experiments to explore 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 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 spanned at 5400 RPM. We evaluated four variations of this disk: spinning at 6400 RPM and 7200 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(a). 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 8b), 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.)

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

Fig. 9. Effects of (a) data layout and (b) mirrored storage class read disk selection policy on
performance.
93% for the cello-user 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 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 article 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 Ruequier wrote the DB benchmark used for the results in Section 3.2.

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

REFERENCES


Received September 1995; revised October 1995; accepted October 1995