Managing Massive Arrays of Idle Disks (MAID) on Linux

Mark King
B.A.(Mod.) Computer Science
Final Year Project  April 2011
Supervisor: Dr. David O'Callaghan

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

______________________________  April 11, 2011
Mark King
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

_____________________________ April 11, 2011
Mark King
Acknowledgements

I would like to thank my supervisor, Dr. O’Callaghan, for his interest and advice during this project; my parents, for their continued support; and my friends, for their encouragement throughout these past weeks and months, especially James, whose thoughtful advice has always been greatly appreciated.

I would also like to thank my classmates in CS2011, as they have made the past four years the most interesting and enjoyable of my life so far, and I could not have asked for a finer group of peers.
“Data is a precious thing and will last longer than the systems themselves.”

– Tim Berners-Lee

“Errors using inadequate data are much less than those using no data at all.”

– Charles Babbage
Abstract

Massive Arrays of Idle Disks (MAID) are a means of making near-line data backup a viable, low-cost alternative to the offline tape backup solutions that are currently the most popular means of data archiving. The concept of MAID has already been studied and bench-marked in the paper ‘The Case for Massive Arrays of Idle Disks (MAID)’[1]. MAID is now available as a pseudo-RAID level on many commercial disk storage controllers[2].

This project aims to bring some of the research in the original MAID paper up to date with today’s state of the art, and to implement MAID in software to run under the GNU/Linux operating system. The design of such a piece of software will be explored, and problems which arose in the implementation are documented along with the solutions that were arrived upon.
Contents

1 Report Layout 4

2 Background 5
   2.1 Introduction ............................................. 5
   2.2 RAID ..................................................... 6
   2.3 MAID ..................................................... 6
   2.4 Tape Storage ............................................. 7
   2.5 Disk Storage ............................................. 8
   2.6 Power Saving Features ................................. 9
      2.6.1 hdparm ................................................ 9
   2.7 The Linux Kernel ......................................... 10
   2.8 Device Mapper ........................................... 10

3 Motivation 12
   3.1 Example Use Cases ....................................... 12
      3.1.1 Lower cost of entry ................................ 12
      3.1.2 Superior latency and throughput ................. 13
      3.1.3 Scope for redundancy ............................ 13
      3.1.4 Testing MAID technology ......................... 13
   3.2 Hypothetical Comparison ............................... 13
      3.2.1 MAID-based solution ............................... 14
      3.2.2 Tape-based system ................................ 14
      3.2.3 Comparison ......................................... 15
   3.3 Software Implementation ............................... 15

4 Design 16
   4.1 Design Goals ........................................... 16
      4.1.1 Power Saving and Latency ......................... 17
List of Figures

2.1 Cost of 1GB of disk storage per year[3] . . . . . . . . . . . . . 8

4.1 Illustration of I/O redirection by the Kernel module, with cache 21
4.2 Illustration of proposed architecture and interaction between
constituent elements . . . . . . . . . . . . . . . . . . . . . . . . . . 23
Chapter 1

Report Layout

This report is laid out into the following discrete chapters: background, motivation, design, implementation and conclusion. ‘Background’ provides relevant information that may be necessary to explain aspects of the rest of the report. ‘Motivation’ focuses on the reasons for pursuing this project. ‘Design’ describes the high-level software architecture that was decided upon, and reasons for this. ‘Implementation’ documents the development process based on the design, and problems that arose during this development. Finally, ‘conclusion’ looks at the outcome of the project, and some interesting possibilities for future work.
Chapter 2

Background

2.1 Introduction

Data storage and archiving are essential features of any system that deals with large volumes of information. Data can easily be lost by way of hardware failure, user error, malicious means and otherwise. Be it user data, research data or otherwise, the possibility of loss of data is something that most projects aim to eliminate.

Modern data storage infrastructure tends towards tiered systems, in which there are several different levels of repositories for the data. The nomenclature is somewhat fuzzy here, so for the purposes of this report, we shall talk about ‘online’, ‘offline’ and ‘near-line’ storage. Online storage is directly accessible by the live system (web application querying a database, large number crunching simulator accessing a dataset, etc.). Offline storage is data which has been archived, and the medium removed from the system. In order to access this data, the medium must be reconnected to its reader by human or robotic means. Near-line data storage is a compromise; the medium is still connected, but is powered down or otherwise not immediately ready for I/O when it is not in use. When it is necessary to access the storage device, it can be electronically woken up from this state by the system.

Data is saved to online storage for quick retrieval – version controlled source code and research data, for example. There is usually some level of redundancy and fault tolerance built into the online storage system, usually by means of a RAID array (see section 2.2). It is often the case that archival
snapshots of data will be taken and put into offline storage for retrieval should they be required after catastrophic failure of the online storage system. Recovery from a medium such as tape can take quite a long time; there are many factors which add up to create quite a high latency in accessing this offline data. We shall explore some of these factors later in this report (see section 2.4).

Data archiving by means of arrays of disks has not seen nearly as much use as offline storage such as tape backup. Two of the chief reasons for this are the heat dissipated by disks and the power consumed by a disk connected to a live system. We shall explore the means by which we might overcome such obstacles and make near-line disk storage a viable alternative to offline tape based systems, but first some background information might prove useful.

2.2 RAID

In reading this document, it may be advantageous to have some knowledge of RAID technology. RAID (Redundant Array of Independent Disks, previously ‘Inexpensive’ Disks) is an umbrella term for data storage schemes that provide improved redundancy by means of an array (i.e. collection) of disks. There are several different types of RAID, but in all forms, data is stored across the disks in the array, making them appear as one disk to the computer’s operating system.

MAID can be seen as a pseudo-RAID level; we wish to present a large collection of disks to the operating system as one contiguous storage device, only we wish to optimise this array for low power consumption and heat dissipation, as opposed to the redundancy RAID provides. Redundancy is not one of our goals here, and can be provided by higher level mechanisms which we need not deal with if we provide a transparent storage medium to the system.

2.3 MAID

Massive Arrays of Idle Disks (MAID) are collections of disks that are used to provide the system with a transparent, contiguous storage medium, with slightly higher latency than a normal disk array, and which are optimised to
consume as little power and dissipate as little heat as possible. We achieve this by means of spinning down the drive – putting it in ‘idle’ or ‘standby’ mode – as often as is possible (which is not to say we constantly request the drive spin down – one of the reasons for this is discussed in section 4.1.1).

Power saving features in hard disks have come a long way over the years, with several different schemes available in modern disks, such as Advanced Power Management (APM) and IDE power modes. We shall exploit some of the power saving features that come as standard in modern drives in the design of our low-power storage solution.

A naive approach might merely allow raw access to the drives in the array and attempt to put the drives back into idle mode after they have been accessed, but there are several means by which we might improve upon this approach. Chief amongst them are the use of one or several drives as a disk cache which can service I/O without waking the storage disks, and the alternative migration scheme, in which we designate some drives as ‘active’ and others as ‘passive’, and aim to have the most frequently accessed data on ‘active’ drives by migrating data across the drives. These approaches are studied in more detail in the design chapter.

2.4 Tape Storage

Magnetic tape storage has been in use for over 50 years, and to this day is a tried and true means of data backup and archiving. Large tape libraries are used to handle data storage and recovery. These tape libraries store the tapes themselves, the robots which load and unload tapes, and the tape drives which actually transfer data to/from the tapes.

Tape libraries store a large number of tapes, read by a variable number of readers, with tapes being changed by several robots – most tape libraries are highly expansible. When data is to be read or written, the tape loading robot first chooses the appropriate tape from the library, loads it into a tape drive and the drive must then seek to the appropriate place on the tape before beginning the I/O operation. Latency of access is increased by several factors: the time it takes for the robot to load the tape, the seek time of the tape drive, and the read / write speed of the drive.

Due to these latency issues, tape storage does sequential access very well, but random access involves high latency. In the case of restoring from an archived backup, time is obviously critical – the system has failed massively...
and has lost all of its data, and as such cannot function until the data is restored – but this data could be stored anywhere in the library and won’t be available until the appropriate tape has been located, loaded, searched and seeked to the correct position.

2.5 Disk Storage

Disk drives are the primary means of online storage. Disks are low-latency, have excellent throughput, high storage densities and are relatively inexpensive. While the cost of tape media has remained fairly stable, the cost of commodity disk drives has been declining for years and continues to do so at an exceptional rate (see Figure 2.1) – a 2TB drive can be purchased for less than €77[4].

![Cost of 1GB of Disk Storage per Year](image)

Figure 2.1: Cost of 1GB of disk storage per year[3]

Tape storage remains more popular than disk storage for data archives due to the always-on nature of disks. When data is archived on tape, it is shelved – consuming no power and producing no heat. Disks, on the other hand, are always on, always drawing power and always generating heat. In a data centre, cooling is a top priority, and power usage can become costly for always-on drives.

On the other hand, if these issues can be sufficiently mitigated, benefits such as vastly improved latency, throughput and the (relative) inexpensiveness of disk storage can be taken advantage of by groups in need of data archiving.
2.6 Power Saving Features

As storage technology has developed, several power saving features and standards have been developed and nearly all drives on the market today support one or more of these features. Advanced Power Management (APM) is one such standard, developed by Microsoft and Intel in 1992[5].

2.6.1 hdparm

‘hdparm’ is a Linux program which allows one to send various ATA commands to disk drives. This includes several power saving commands, and commands which allow us to inspect the power state of the drive – these are key to the functioning of software MAID. A listing of the commands follows, taken from the hdparm manual page[6].

-B ⟨value⟩
This option sets the Advanced Power Management level. A low value means aggressive power management and a high value means better performance. Values 1–127 permit spin-down, while 128–254 do not, and 255 disables APM entirely. Our system will use a value of 1 to provide aggressive power management.

-C
This option requests the current IDE power mode from the drive. This may be one of ‘active/idle’ – normal operation, ‘standby’ – low power mode in which the drive has spun down, ‘sleeping’ – lowest power mode in which the drive is completely shut down, or ‘unknown’ – the drive does not support the command.

-S
This option puts the drive into the idle (spun-down) state, and also sets the timeout which determines how long the drive waits without disk activity before spinning down automatically.

-y
This option instructs the drive to enter the idle or standby state, which spins down the disk. The time it takes to spin up from this state is a matter of a few seconds.
This option puts the drive into the sleep state, which is the lowest power level available. The drive will be completely powered down apart from the controlling logic board that allows the drive to wake up again once it receives requests for I/O.

2.7 The Linux Kernel

The Linux kernel is a free and open source operating system kernel, initially released in 1991. It is written in C, and contributed to by thousands of developers worldwide. It is modular in design, and kernel modules can be linked in at run time. There are several interfaces provided which can be used to write kernel drivers. In an effort to reduce latency and increase responsiveness, modern Linux kernels are preemptible – the kernel itself is scheduled, much like userspace processes are – only the kernel scheduler itself is safe from preemption. This is true on single core processors as well as on multicore machines, and introduces the need to design kernel drivers robustly, in a thread-safe manner. Another constraint which must be considered when writing kernel modules is memory usage.

The kernel source code is maintained using a version control system called Git, written specifically for this purpose, but now widely adopted by the open source community. Snapshots of the kernel source code can be acquired from http://kernel.org/, and patches may be contributed by anyone.

There are two main types of device drivers in the kernel – char devices and block devices. Block devices are those which can be addressed by sector number, with sectors usually composed of 512 bytes. We wish to implement our software MAID system as a virtual block device, to allow userspace code to access MAID storage in an otherwise transparent way.

2.8 Device Mapper

Device mapper[7] is a framework for creating and managing virtual block devices and interjecting I/O requests to said device to change, redirect or otherwise programmatically modify I/O. Developers can write a so-called ‘target’, which will hook into the interface provided by device mapper. These targets provide various different types of virtual block device, each of which can be
used for different purposes. Device mapper consists of several components:

**Device mapper kernel module (dm-mod)**
The kernel module handles some of the lower level routines common to all targets. It also includes some of the simpler targets, built-in, such as dm-linear – creates one large contiguous virtual block device from several smaller ones, dm-zero – returns ‘0’ on reads and silently drops writes, and dm-error – generates I/O errors on reads and writes, mainly used for testing purposes.

**Device mapper module interface**
This is an interface for hooking into the device mapper kernel module, and registering your module as a device mapper target that can be called to create virtual devices.

**Device mapper userspace library (libdevmapper)**
The userspace library enables user programs to query the device mapper kernel module for information on a device, set up new virtual devices, remove devices, suspend devices and resume devices.

**dmsetup**
This is a userspace program which exposes most of the commands which allow interaction with the device mapper kernel module. It can be used to create, destroy, suspend, resume, re-map, and otherwise query virtual devices owned by device mapper.
Chapter 3

Motivation

In this chapter, we shall explore some of the reasons we would wish to create a software solution to MAID, as well as update some of the research pursued in the original MAID paper.

3.1 Example Use Cases

Presented in this section are some examples of scenarios in which a MAID system has obvious advantages over a traditional tape based archive.

3.1.1 Lower cost of entry

Large tape libraries (of the magnitude of several hundred terabytes of storage) can cost upwards of tens of thousands of euros (see section 3.2.2 for an example system). Adding to that the cost of several tape drives, several hundred tapes and expensive proprietary backup software, it can be seen that the cost of putting such a storage solution together is very high. Such an investment may not be an option for a small startup business, but data archiving may still be desired for or even essential to the operation of the business. The cost of a commercial storage controller implementing MAID will be less than that of a tape library, but cheaper still would be a consumer level PC running our software MAID system under Linux, with an array of regular commodity hard disks.
3.1.2 Superior latency and throughput

Possibly the strongest motivation for MAID is the hugely increased throughput and latency that it offers. As opposed to offline tape backup, MAID is nearline – the disks can be woken up in a matter of seconds and once woken, provide the same throughput of an online storage solution. Tape library latency is measured on the order of minutes, instead of the seconds a MAID takes to wake. We shall explore an example comparison in section 3.2.3 for some concrete figures.

3.1.3 Scope for redundancy

Due to the far lower cost of the hardware required to build MAIDs as opposed to disk libraries, there is more scope for redundant storage – as a basic example, two identical MAIDs storing the same data. If one fails, the other acts as a fallback, and the failed hardware can be cheaply replaced. Tape storage often lacks redundancy as it is so expensive. Once a data snapshot is shelved on tape, there is no guarantee that snapshot has survived intact. It is unusual for snapshots to be stored redundantly – especially for large datasets – due to the cost of the tapes themselves.

3.1.4 Testing MAID technology

If a group is considering investing in a dedicated hardware storage controller capable of managing MAID, but want to try running a MAID first, the software solution proposed here allows them to trial-run such a system with negligible cost and hopefully comparative performance.

3.2 Hypothetical Comparison

Imagine a system is being designed which requires 12TB of archival storage. This section shall present two hypothetical storage solutions for such a purpose – one a tape library, one a MAID – and compare the features of each.
3.2.1 MAID-based solution

The hypothetical 12TB cache-based MAID (see section 4.2.1) will use six 2TB Samsung Spinpoint drives for storage and one 2TB Samsung Spinpoint drive as the cache disk. These disks are rated as follows, according to their data sheet[4].

<table>
<thead>
<tr>
<th>Throughput</th>
<th>300MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spin-up time</td>
<td>13 seconds</td>
</tr>
<tr>
<td>Average seek time</td>
<td>8.9ms</td>
</tr>
<tr>
<td>Response latency</td>
<td>5.52ms</td>
</tr>
</tbody>
</table>

At the time of writing, such a disk array might be purchased at the following prices.

\[
7 \times €77 - 2TB Samsung Spinpoint drives[4] \\
€539
\]

3.2.2 Tape-based system

The 12TB tape archive will be composed of four HP MSL5030 tape libraries, each of which holds a maximum of thirty 100GB capacity tapes to provide 3TB of storage each. Each library houses two drives, with the following ratings provided from the data sheets (the library[8] and the tapes[9]).

<table>
<thead>
<tr>
<th>Throughput</th>
<th>15MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. rewind time</td>
<td>80 seconds</td>
</tr>
<tr>
<td>Average access time</td>
<td>50 seconds</td>
</tr>
</tbody>
</table>

At the time of writing, such a system might be purchased at the following prices.

\[
4 \times €3,865 - HP MSL5030 Library[10] \\
+120 \times €27 - Dell LTO-3 100GB Tape Cartridges[10] \\
€18,665
\]
### 3.2.3 Comparison

The following table shows how these hypothetical storage solutions compare in a scenario where they are being read from at sustained maximum throughput. In the tape library, half the drives are assumed to be performing I/O, and the other half seeking or loading tapes at any given time. The cache disk in the MAID is not included as providing throughput. Data would be copied into the cache as all the disks serviced their I/O requests, but it would be immediately dropped from the cache due to the unusually high amount of I/O being performed, and would theoretically receive a negligible amount of hits.

<table>
<thead>
<tr>
<th></th>
<th>Tape Library</th>
<th>MAID</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max. Throughput</strong></td>
<td>60MBps</td>
<td>1800MBps</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>90 seconds (from tape load)</td>
<td>13.01442 seconds (from idle)</td>
</tr>
<tr>
<td><strong>Est. build cost</strong></td>
<td>€18,655</td>
<td>€539</td>
</tr>
</tbody>
</table>

This comparison illustrates the clear advantages of a MAID system over a tape library – increased throughput, reduced latency and a less expensive build cost. Unfortunately, the table does not show power consumption figures. The required hardware was not available for testing during the writing of this report, and as such no concrete figures could be produced. The hard drives are rated for 6.3 Watts when performing I/O, and 1 Watt when idle. The tape libraries are rated for 200 Watts at peak usage. Unfortunately, these figures mean very little to the comparison, as the hardware will have a varying load, and power consumption will vary from the rated Wattage.

### 3.3 Software Implementation

MAID technology has already been implemented in some commercial disk storage controllers, such as Nexsan’s AutoMAID[2]. This project aims to implement MAID technology fully in software, to remove the cost of such controllers to small groups interested in running archival storage. This continues the trend of RAID levels being implemented in the md and lvm drivers for Linux. While these commercial storage controllers have large research and development funding to drive a very high level of performance, it is hoped that the software suite created at the end of this project will offer a very comparable performance.
Chapter 4

Design

A large portion of time on this project was spent in research of technologies and planning of the design of the software. Several key design goals were identified, and these led to design choices being made as detailed below.

4.1 Design Goals

The goal of MAID technology is to take advantage of hard disks’ advantages in latency and throughput, whilst mitigating the problems of heat disapparation and power consumption. It is also desirable that the system is stable, robust and reliable. To those ends, the following design goals were arrived upon.

Power saving
Without meeting this goal, a MAID is useless for data archiving. MAID is optimised with power saving as the chief and foremost concern. Every design decision made must take this into account.

Latency
Latency is a huge motivating factor in the use of MAIDs. The MAID controller software must be as fast and efficient as possible so as to negatively impact this as little as possible. This becomes a challenge when parallel access (thread-safety) and cache coherency lead to locking of data structures and decreased performance.

Throughput
Throughput, much like latency, needs to suffer as little impact as pos-
sible when considering the use of a plain hard drive versus a disk in a MAID managed by the software being designed for this project.

**Data Integrity**
A data archive is pointless if the data it contains is not correct and intact. Data integrity is in fact a design requirement, rather than a goal. Several methods can be used to ensure data integrity, at both the file-system and block device levels.

**Hardware Reliability**
Disks in a MAID are particularly susceptible to wear, and so it becomes necessary to monitor disk health closely to ensure the system has not suffered any failures.

### 4.1.1 Power Saving and Latency
The software will make use of the power saving features available in nearly all modern drives in order to put them into standby (also referred to as idle) or sleep mode. When in one of these low-power modes, there will be a relatively large latency when it is required to wake the disks to service some I/O request – this waking process occurs automatically when the disk controller receives an I/O request and the disk is seen to be idling. Due to this latency, we do not want to prematurely idle the disks if there is a fair chance that more I/O is queued to this disk, but rather perform some intelligent throttling on how often disks are put to sleep. To this end, the software will respect a timeout between the time when a disk is woken up and when it should be put back to sleep, depending on how long the disk remained idle previously. Information must be kept about how long each disk has been idle since last waking and the last timeout factor applied in powering down each disk in question.

The following formula will control the timeout factor used in powering down disks.

\[
T(t, s) = \begin{cases} 
1 \text{ minute} & \text{if } s \geq 30 \\
t \text{ minutes} & \text{if } t \times 2 < s < 30 \\
2t \text{ minutes} & \text{otherwise}
\end{cases}
\]

where \( t \) is the previous timeout used and \( s \) is the number of minutes the drive previously remained idle.
The cases in this formula are designed to return a fair timeout based on how successful the previous timeout was at letting the disk finish its I/O. From experimentation, it does not appear that receiving a spin-down request whilst performing a lot of I/O will interrupt that I/O – the spin-down is aborted as soon as the next request comes in. The case that this throttling prevents is where the disk has just barely managed to return to idle before receiving more I/O, and enters an unfortunate loop of spinning down only to spin up relatively soon afterwards. A timeout of 30 minutes is introduced to reset the timeout, as MAID would not expect this amount of sustained I/O missing the cache. If it occurs and the timeout resets prematurely, the timeout will grow back fast enough to quickly compensate. The growth rate is another element that may require tweaking – a factor of two may prove to increase the timeout too aggressively or not aggressively enough. It may be desirable to make these factors configurable, but hopefully these defaults are sane enough to provide adequate performance in most scenarios.

4.1.2 Throughput

It is of utmost importance that the code which services I/O requests to the virtual MAID device and subsequently performs the real I/O on the disk array is highly performant. As this code will affect every request, it has a direct effect upon throughput\(^1\). I/O traffic may be broken down into many discrete sub-requests in the kernel before being handled by the code that manages the MAID, and any slight performance penalty in this code will be multiplied perhaps a hundred- or thousand-fold in worst case scenarios. The logical consequence of this issue is that any optimisations made to this code – even if of small benefit in the scope of a single I/O request – will show real performance benefits in large scale.

4.1.3 Data Integrity and Redundancy

It is vital that while reducing power consumption and heat disappation, the MAID software maintains the integrity of the data it reads and writes. The code must be written with this in mind, as an incorrect mapping from the virtual MAID device to the physical storage disks is a real possibility. Every

\(^1\)It will have an effect upon latency also, but this will not be nearly as apparent.
possible scenario in which the software would read or write the wrong data must be tested and resolved.

Aside from data corruption due to bugs in the MAID code, problems with data integrity may also occur naturally at different levels of abstraction in our system – due to disk wear or failure, current spikes, bugs in disk failure and etc. Efforts can be made at file-system, storage architecture and block device levels to ensure the integrity of the archived data stored on the MAID.

At the file-system level

There are multiple means to check for errors in data integrity at this level. One example is the use of a filesystem designed to maintain data integrity, such as ZFS[11], which can sit on top of the virtual device the MAID software will present.

Another such example is keeping track of data integrity by means of noting the hash of each file in storage before it is stored and periodically comparing this with the freshly-computed hash of each file. Any discrepancy here will point to data corruption. This would need to be implemented on a higher level than the software presented here deals with – it would need to preempt file creation and modification at a file-system level, rather than block level I/O which is what the software must interject in to create a virtual MAID device.

4.1.4 At the storage architecture level

When designing a data archiving solution, the system architect may desire redundancy on top of the features MAID provides. One simple way to do this is to create two or more MAIDs on a system and use the virtual devices these MAIDs provide in a software RAID setup to provide the required redundancy. An alternative is to have several separate MAID systems, and mirror data across them.

At the block device level

At this level, we can use any of several widely available tools to perform surface scans for bad blocks and similar. One such tool is the Linux program ‘badblocks’. This tool can be used to generate reports of bad blocks on any
drive. The software MAID system will use this tool when generating periodic reports, so as to alert administrators of any possible data corruption issues.

4.1.5 Drive Health

Disks in a MAID are particularly susceptible to wear. The primary cause of wear to hard disks is the spin-up/spin-down process. In a MAID, this happens with far more frequency than under normal use. Drive heads may also become stuck if left stationary for too long, rendering the disk unusable.

In order to monitor drive health and prevent parked heads becoming stuck, the MAID software will periodically spin up each of the disks in the MAID and run health checks on them. A popular tool for monitoring drive health is smartd, which considers a variety of factors which may lead to disk failure – acquired from drives implementing S.M.A.R.T – and can generate reports based on these factors, warning of imminent disk failure if it seems to be indicated. This will allow administrators to recover the data on the failing drive and replace it before failure.

4.2 Minimising Disk Wake-Up

Rather than naively passing I/O requests directly to disks in the MAID – thus waking them for every I/O request – and storing data linearly across all disks – thus allowing each disk to be woken as often as any other – steps can be taken to reduce the frequency at which disks are woken to service I/O requests. In the original MAID paper, two techniques were considered for reducing the amount of wake-ups: using a cache disk, and migrating data between disks based on frequency of access. These techniques are described below.

4.2.1 Cache-based MAID

In cache-based MAID, one or more of the disks in the MAID are designated as a cache disk and the rest are used as storage disks. The cache disk remains much more active than the storage disks. When an I/O request is received, the cache is queried to see if it can service the request. If not, the storage disk containing the requested data must be woken and service the request.
This data is then added to the cache for subsequent requests, and the oldest element in the cache is forced out.

It was decided that a write-through cache should be used. In typical archival use, data will be written once and often will never be fetched again, or else will be fetched in the far future. As such, we will get very few write hits to the cache. One case for using a write-back cache, where writes are not immediately committed to the storage disk, is that disk wake-ups could be more efficiently timed; that is, when a drive is woken for health checks and etc., any writes in the cache destined for that disk could take such an opportunity to write through. If there is not much pressure on the cache, this could result in disks being woken less often, but would be most beneficial when a large cache is available. Priorities could be implemented to adjust whether read entries or write entries are dropped or flushed from the cache first. Due to the wish to get a working version of the software developed as quickly as possible, it was decided that a write-through cache would be initially implemented, for sake of speed.

As one or more disks are explicitly used as a cache and not for storage, this type of MAID will require more raw storage to function than it provides. This would be considered a drawback by some, but the investment in a cache disk will hopefully pay for itself quickly by reducing heat disappation and power consumption.
4.2.2 Migration-based MAID

In migration-based MAID, data is migrated (that is, relocated and swapped with other data) around the disk, so that popular data moves towards ‘active’ disks, and unpopular data ends up on ‘passive’ disks, which can spin down for much longer. A nice feature of this technique is that the entire array of disks is used as storage, but it was decided that the software presented here would not use this approach as it involves keeping a huge amount of meta-data (for every single chunk of data in the array, so that the virtual addresses of the logical block device can be properly mapped to the real disks), and then sorting this huge collection of data to find the optimal layout for the data. Such a large amount of computation could not be done in the code which handles I/O redirection, as in this code, speed is critical. The migration would have to be performed by a daemon process at regular intervals, and the MAID would have to be taken offline every time this migration process was run, to prevent data loss or improper access. Any I/O that was issued during this migration process would have to wait for it to finish, and thus there would be massive latency in such cases.

4.2.3 Hybrid Approach

A theoretical hybrid cache/migration based MAID system could be created. Although this is not discussed in the original MAID paper, it may be possible to reap the benefits of both approaches simultaneously; a cache would be put in place to stop requests hitting the storage disks, but also, the data on the storage disks could be moved so as to order it by popularity across the disks. Such an undertaking would be extremely complicated, and would have massive overhead in terms of meta-data needed to be stored by the kernel. As memory is a commodity in the Linux kernel, it was chosen to avoid this approach, but it might make for interesting future work if a viable solution is found to cut down overhead.

4.3 Proposed Architecture

The software architecture arrived upon consists of two parts: a userspace MAID daemon (titled maidd), and a kernel module based on the device mapper framework (titled dm-maid). Each handles separate parts of the
functionality of the system, and they can communicate data to each other in order to perform joint functions.

4.3.1 Monitoring daemon (maidd)

The monitoring daemon will primarily be responsible for power-managing the disks. It will intelligently instruct the disks to sleep based on the timeout algorithm mentioned previously.

In addition to power management, the daemon will periodically spin up the disks so as to avoid the disk heads becoming stuck. It will take this opportunity to run health checks over the disks, including using the badblocks
and smartd tools described above. These reports will include data concerning popular data, retrieved from the kernel module, and will be stored in a log to be read by system administrators.

4.3.2 Linux kernel module (dm-maid)

The kernel module will be based on the device mapper framework. It will use a write-through cache to service I/O when possible. Cache-based MAID was chosen over migration-based MAID for several reasons, including feasibility for implementation within the time constraints to this project, issues arising from the large amount of overhead involved in storing meta-data for any reasonably sized MAID, latency involved in performing migration, and the added level of control of power levels introduced when using a cache.

The kernel module resolves virtual addresses on the logical device into real addresses for disks in the physical array. If a read request can be serviced from the cache, the request will be redirected to the appropriate address on the cache disk. Otherwise, the kernel module will act as a proxy for the I/O request, perform it, and add the relevant data to the cache before filling the original request with the retrieved data. Write requests will be redirected directly to the storage disks.

4.3.3 Communication between daemon and kernel module

Information concerning I/O performance and the contents of the cache will be retrieved from dm-maid by the monitoring daemon by means of IOCTLs – small pseudo-system calls that are implemented by device drivers, such as the kernel module. This information should prove useful for monitoring performance of the MAID.
Chapter 5
Implementation

This chapter documents the development and testing of the MAID controlling software. Issues that arose during the development process are raised and their resolutions discussed. Choices made during implementation are described and evaluated.

It was apparent early on in the project that it would be necessary to create two discrete and complimentary pieces of software to perform the functions required of a software MAID implementation: one would run in the Linux kernel space, acting as an I/O controller, redirecting and modifying requests as they came in; the second would be a userspace daemon which would handle power management of the disks and perform other miscellaneous maintenance routines.

5.1 Creating the MAID Kernel Module

As it was the author’s first experience writing kernel code, it was necessary to do much research in the area. The Linux kernel is a very different programming environment to regular userspace programming. Pre-emption, parallel access, memory restrictions, latency issues and the lack of the usual standard C library made the creation and development of dm-maid a very interesting challenge. The Linux Documentation Project had several good articles about kernel development, and LWN publish a PDF titled ‘Linux Device Drivers, Third Edition’ which proved to be an invaluable resource for information about Linux kernel development[12].

The comparison between MAID and RAID had been drawn early on in
the lifetime of the project, and this pointed to the possibility of modifying one of the implementations of RAID controller software in the kernel to act as a software MAID controller. Two popular pieces of software were found which provided RAID-like functionality: md, which is most popular for software RAID and device-mapper, which is the basis for lvm2, popular for creating and managing logical disks. As md is an older piece of software, and device-mapper was found to be created specifically to allow programmers to create logical devices backed by real physical disks and control block I/O to such devices – without having to deal with much of the boilerplate code that goes into such implementations, and with a suite of tools for easy setup of such virtual devices – device mapper was chosen as the basis upon which to create software MAID.

5.1.1 Device Mapper Development

The process of developing the device mapper target (named ‘dm-maid’) was one of much trial and error and reverse engineering of already available targets. Literally no documentation exists dealing with the creation of a new device mapper target, which left the author in the proverbial deep end in terms of learning curve.

Device mapper development is primarily handled by Red Hat, Inc. It was discovered that there exists an IRC channel for device mapper development, and the author attempted to reach developers there for some advice. Two problems became apparent from visiting this IRC channel: the developers were very busy and could take many hours or sometimes days to offer any response, and the developers who frequent the channel are ostensibly all developers of the userspace library, but have little or no experience with the code base of the kernel device mapper interface – the code which must be extended to create new device mapper targets. As such, this avenue of research proved a dead end.

The development process – with no documentation or other means of getting a real overview of a device mapper target implementation – consisted primarily of experimentation with and modification of existing device mapper targets. This would lead to an unfortunate circumstance later in the project, in which some of the code from the dm-linear target was modified for use in dm-maid, but proved inefficient for use due to the meta-data that dm-maid must store for controlling its cache (see section 5.1.2). After much effort was put into understanding the significance and use of each function
used in the considered existing targets, an initial ‘dumb’ version of dm-maid was developed, which merely mapped I/O from the logical device onto real storage, without any caching. After this was seen to be working reliably, more features could be added on, iteratively increasing the functionality of the software.

5.1.2 Cache issues

The device mapper targets initially used as reference implementations to be derived and extended to create dm-maid dealt exclusively with redirecting I/O, and never dealt with the case of proxying I/O so as to be able to inspect the results before returning them. Perhaps because of this, certain issues with the cache were initially overlooked in the design of dm-maid.

**Proxying I/O**

When an I/O request gets to the dm-maid code, it has already worked its way through an I/O queue to be serviced. By proxying this, and effectively re-submitting the I/O request, the request must sit through another queue. In periods of heavy I/O, this could lead to latency issues, which was worrying. A workaround for this has not been found yet, as this functionality is essential to the operation of the cache, but it is considered that since increase in latency will only be on the order of milliseconds, it is a fair trade-off for the power reduction the cache will allow.

**Cache page size**

Another factor initially overlooked which became apparent towards the end of the development process is the size of pages in the cache. As mentioned, the code which handles the data in the cache was written to manage individual sectors of 512 bytes. During testing, virtual disks were used, as not enough physical disks were available to put together a large enough MAID. An arbitrary decision was to make these drives 50MB in size – the size was not thought to matter, and smaller virtual disks could be transferred between computers more easily.

Towards the end of the project, these virtual disks were replaced with much larger disks several gigabytes in size. It was found that due to the massive number of sectors on these drives, when trying to allocate a data
structure to store meta-data for each sector, the amount of memory requested of the kernel could not be provided, and so the device mapper target aborted initialisation with an insufficient memory error code.

The allocation of memory in the kernel was researched more thoroughly at this point, and was found to be a very complex area. The kernel preserves a very small area of memory for its own use, and this is not sufficient for dm-maid’s needs. It is thought that requesting memory by means of the vmalloc system call as opposed to the kmalloc system call may solve this problem, due to the different pools of memory each allocates from; kmalloc attempts to allocate physically contiguous chunks of memory, whereas vmalloc tries to allocate contiguous virtual memory addresses. This solution has been tested and appears to be the case, but more rigorous testing is necessary.

An alternative approach to reduce the overhead in storing meta-data about the cache is to treat the cache as being composed of chunks larger than one sector in size. It is proposed to rewrite the cache management code to use arbitrarily defined chunk sizes – initially a size of 64KB was considered – so that the performance of the cache can be fine tuned for individual use cases. This feature may have an added benefit, by taking advantage of locality of reference. It is likely that when I/O is requested for some part of the disk, more I/O requests will arrive for nearby areas of the disk, due to the nature of file accesses. Consider streaming a video from disk, for example, or compressing a large text file. If we are pulling slightly more data than was requested (due to chunk size), it is reasonably likely that this pre-fetched data will be requested by a subsequent I/O request soon after the original. As this data is already in the cache, performance of our MAID increases.

Unfortunately, there was insufficient time to perform such the huge overhaul and redesign of the code necessary to safely and stably implement this ‘chunking’ behaviour, and while initial steps were taken to develop this functionality, it is not currently in working order. It is hoped that the rewrite will be complete soon after the submission of this report, and the latest sources for dm-maid can be downloaded from the project’s website if this ‘chunking’ functionality is desired.

**Cache coherency**

It became apparent that without proper measures, I/O requests redirected to the cache could potentially return the wrong data if that particular cache
entry had been dropped and replaced with a new entry. As I/O is completely untraceable once being redirected and leaving the dm-maid code, we cannot know when any given request will be finally read from the cache. Due to this, we need to proxy requests to the cache, as opposed to redirecting them. When a request generates a cache hit, we atomically update the meta-data for that entry to increase the number of references currently held to that entry. When the proxied I/O request is seen to finish, the reference count is atomically detrimented. An entry with a non-zero number of references will never be replaced in the cache, to ensure reliable reading of data stored in the MAID.

How data leaves the cache

When a new entry is to be added to the cache table, we must drop something previously held in the cache. On initialisation, every cache entry will have an UNUSED flag set, so that the cache can be populated immediately. When it becomes full during normal operation, we have to choose an entry to drop. This is handled by looping over the cache table searching for entries with a reference count of zero. Upon finding such an entry, the entry in question is atomically updated to describe the new data in the cache.

One possible issue that could arise here is an unfair wait when searching for unreferenced entries. It may happen by chance that a search may take an unreasonable amount of time to find an unreferenced entry, as new I/O requests may prevent any entry’s reference count from dropping to zero. A proposed solution to this is to tag entries with a ‘logically deleted’ flag, which when checking cache entries would be taken to mean ‘ignore this cache entry’, and so the reference count would never increase but would strictly decrease as I/O requests for that entry finish.

5.1.3 Testing methodology

Due to the nature of the software – a module running as part of the operating system kernel – it was necessary to use virtual machines to test dm-maid as it was developed. If code changes introduced instability that led to a crash, the operating system itself could fail, requiring the development system to be rebooted to recover. Virtual machines can more quickly be restored to a working state ‘snapshot’, and so were perfect for kernel development and testing.
5.2 Creating the Monitoring Daemon

The monitoring daemon did not present nearly as much of a programming challenge as the kernel module due to the author’s work in systems administration, and by way of this, decent body of experience in scripting. The development process consisted mainly of tweaking the design of the algorithms presented in the previous chapter very slightly, and addressing bugs as they cropped up.

5.2.1 Managing power levels

The daemon makes system calls to run the hdparm program with its various flags to control power levels of disks in the MAID. It also polls the power state of the disks in order to calculate the timeout period between issuing sleep commands to disks.

5.2.2 Problems generating S.M.A.R.T. reports

Unfortunately, smartd is a daemon, and as such, will access the disks at inopportune times (while they are spun down), and can’t be controlled by the monitoring daemon. The resolution to this is to port the functionality of smartd into the monitoring daemon entirely, but this will be a long and tedious process, and was not in the scope of this project due to the large portion of time that would have to have been allocated to this feature, but in the scope of future work, this is very possible and is a desirable feature.

5.2.3 Support of different power saving features in hardware

Initial testing of the monitoring daemon on virtual machine infrastructure proved futile as the virtual disks responded to power management commands with error stating that power management was not implemented. Of the physical disks that have been tested, it is found that disks from different manufacturers – and even different brands of disk from the same manufacturer – will support a varied subset of the possible power management commands available through hdparm. Some did not support checking current power status, some reported standby as active/idle and many seemed to report sleep – the lowest power mode – as standby. Also, of the several means
available for powering down drives, each drive implements and supports a different subset. This lead to the decision to make the monitoring daemon try as many techniques as possible for managing disk power levels, and the hope that most modern drives respect at least one of the commands issued. It is not thought that there is any negative result produced by using this ‘try all’ approach.
Chapter 6

Conclusion

6.1 Outcome

This project produced a Linux kernel module and monitoring daemon script that allow the creation of a software-controlled MAID under the GNU/Linux operating system. Although the code is a first revision, it covers many of the desired features from the initial design. Work can be taken to improve, optimise and stabilise this code, and eventually it is hoped that the code base will become mature enough to gain use in the technology industry. Such real-world use would generate reports of how to further optimise and improve the software, and further increase its usefulness.

6.2 Future Work

In this section, some of the interesting extensions to the project that have occurred to the author during its lifetime, but which were outside the scope of the initial implementation, are discussed.

6.2.1 Write-back cache

As mentioned in the design section, a write-back cache might allow for finer control of the waking of the disks. This would require much more code to handle the interaction between the monitoring daemon and kernel module.
6.2.2 Dual-speed drives

New hard drives have begun to appear on the market with a new power saving feature; they may operate at either of two speeds, high-RPM (Revs Per Minute - the speed at which a disk drive spins) and low-RPM. This dual-speed ability can be exploited to decrease power usage. Perhaps when drives have had some recent I/O activity, we don’t put them into standby mode immediately, and rather more aggressively set them to the lower RPM speed. This would allow us to put the drive into the lower RPM mode more often than we could idle it, as the latency involved in going from low-RPM to high-RPM is much less severe than going from idle to powered-up.

The monitoring daemon could be extended to recognise these dual-speed drives, and use them to increase power savings.

6.2.3 Latency improvements

Much work has been put into ensuring that the MAID cache remains coherent throughout operation and avoids race-conditions, but this comes with some penalties in the form of blocking locks and atomic updates. While dm-maid has been designed with low-latency as one of its chief goals, as with any software project, it could benefit from further efforts in this area.

6.2.4 Rotating the cache disk

The cache disk is the most active disk in the array. As we have noted, the biggest factor contributing to mechanical failure of disks in a MAID array is the large number of times the disk transitions from idle to active. Storage disks in the array are spun up nearly every time they have to be accessed, and this leads to wear. If we rotate the responsibility of being the cache disk over disks in the array, we can more evenly distribute this wear, giving each disk a period of high usage, and low-wear, as opposed to the current system, where one disk is in constant use and the rest are subject to spin-wear.

To achieve this, the array first needs to be taken offline. Once it is offline, we copy the data over from the storage disk to the cache disk, and then swap their roles. The data on the cache does not need to be copied to the storage disk, as it is more efficient to simply allow the empty cache to populate naturally.
Bibliography


Appendix A

CD

Please find the code bundle CD attached to this page.