Distributed MD

André Brinkmann, Sascha Effert,
Michael Heidebuer, Mario Vodisek
University of Paderborn, Germany
Email: brinkman@hni.upb.de

Abstract

Data has become the most valuable asset for many companies; loosing important data can cause companies to fail quite immediately. The protection of data inside storage systems is mostly achieved by using a RAID scheme that adds redundant data to user data, enabling recovery from single or multiple disk failures. This protection against data loss in case of a disk failure can be achieved either by dedicated hardware or a software RAID solution.

One major advantage of software RAID is that it comes for free as a built-in functionality in many operating systems like Linux or Microsoft Windows. The drawback of the built-in functionality is that it is not suited to run in multiple server environments; synchronization and recovery processes can be corrupted if more than a single server is allowed to access a software RAID volume.

In this paper, we present an enhancement for the Linux *md*-driver that enables a consistent usage of RAID in multiple server environments. Based on the V:DRIVE virtualization environment, RAID volumes can be consistently synchronized and recovered even in distributed environments. Besides the architectural concepts, we present measurements that indicate the viability of this enhanced, distributed version of md.

1 Introduction

Data growth has become a major challenge in IT administration, requiring the management of hundreds of storage systems inside single environments. A first step to enable the management of such a large number of disks has been taken in 1988 by Patterson, Gibson, and Katz [10]. They introduced five different RAID levels to overcome the limitations that have been imposed by a single, large, and expensive disk. These single disks did not only lack capacity scaling requirements but also were not able to guarantee the required availability and reliability. RAID has been and is an important research filed, see e.g. [4] [6] [13] [14]. RAID solutions can either be implemented in Hardware or as a software add-on to the underlying operating system.

One major advantage of software RAID is that it comes for free as a built-in functionality in many operating systems like Linux or Microsoft Windows. The drawback of the built-in functionality is that it is not suited to run in multiple server environments; synchronization and recovery processes can be corrupted if more than a single server is allowed to access a software RAID volume.

Software RAID solutions have become an interesting objective for research on storage performance and storage reliability. Brown et al. have introduced availability benchmarks for RAID environments and have presented reconstruction measurements on Linux, Sun Solaris, and Windows 2000 environments [3]. Besides others, they have outlined the trade-off between a short system reconstruction time and the system performance during reconstruction.

Staelin has compared different software RAID implementations for Linux and Windows XP [12]. He has shown that the Linux md driver for the kernel version 2.4 is significantly slower than its Windows XP counterpart and has proposed possible improvements on Linux kernel and block layer design.

Different distributed software RAID environments that enable the concurrent access from multiple servers have been implemented inside distributed files system [7] [9]. Due to Shinkai et al., the implementation inside file systems overcomes problems with distributed locking, recovery and caching. Similar results have been presented in [8], evaluating software RAID file systems based on simulations.

Inside this paper we present a distributed RAID approach that decouples block level driver functionality from the tasks of a distributed file systems. This enables a simple set up of distributed RAID functionality that is based on standard RAID drivers developed for single computer environments. Based on an implementation inside the V:DRIVE storage management environment [1],

we present measurement results for synchronization and recovery in distributed RAID environments.

The outline of the paper is as follows: In section 2, we will give a short introduction into the V:DRIVE storage management environment. In section 3, we present the fundamental differences between building software RAID mirrors in single server environments or in a distributed environment. The focus is on the synchronization process after setting up a new mirrored volume. We will describe the integration of the standard md RAID solution into the V:DRIVE storage management environment. Measurement results for this coupling are presented in section 4. Inside this paper, we compare the performance before, during, and after synchronization and recovery. Furthermore, we will present comparison results between the original version of the md tools and the distributed version.

2 V:DRIVE storage management environment

Inside this section we give a brief introduction into the V:DRIVE storage virtualization environment for Linux 2.4 and Linux 2.6. This introduction is restricted to the properties of V:DRIVE which are required for the understanding of this paper. For a more detailed description of V:DRIVE, see e.g. [1].

In V:DRIVE, physical volumes are grouped in storage pools. These storage pools are not accessed directly, but by the abstract concept of virtual volumes which are exported to the accessing servers. The properties of a virtual volume have not to be related to the properties of the underlying storage pool. It is possible to create a virtual volume with a capacity much bigger than the capacity of the storage pool, unless the used capacity of the set of virtual volumes does not exceed the physically available capacity of the storage pool. Each virtual volume can be concurrently used by an arbitrary number of servers.

The capacity of each disk in a storage pool is partitioned into minimum sized units of contiguous data blocks, so called *extents*. The extents are distributed among the storage devices according to the randomized *Share* strategy which is able to guarantee an almost optimal distribution of the data blocks across all participating disks in a storage pool (see [2]). The typical size of an extent varies between 4 MByte and 512 MByte.

The core component of V:DRIVE is a clustered metadata appliance that stores and distributes information about the SAN environment. This information includes the physical volumes, the storage pools and virtual volumes, and the set of extents which are already assigned to the different virtual volumes (see Fig. 1).

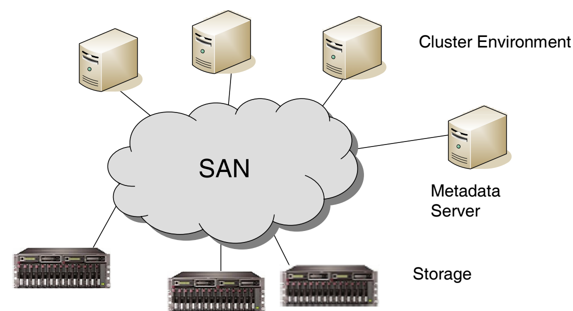


Figure 1: Distributed environment based on V:DRIVE.

Inside the V:DRIVE environment, each Linux server is running a small driver module that presents its virtual volumes to the host operating system. Each time the server accesses an address of a virtual volume that belongs to a new extent, the server has to send a request for the location of the extent to the metadata appliance. The access is delayed until the reception of a valid extent location answer from the metadata appliance. To speed up following request to the extent, the extent location is stored inside an extent cache.

Communication between the metadata appliance and server is done through standard TCP/IP sockets. Both, metadata appliance and server are listening at defined ports for communication requests.

3 Distributed Synchronization Protocols

This section gives an overview about the architectural changes between the single server md-environment inside Linux 2.4 and Linux 2.6 and an environment that can support an arbitrary number of servers working together on a concurrently accessed volume. After briefly presenting synchronization and recovery for single server environments, we will present the distributed synchronization protocols and the required architectural changes inside the V:DRIVE environment. The proposed changes can be also integrated directly into the md driver environment.

Inside this section we only handle a simple RAID 1 scheme; all statements can be generalized to other RAID levels. We will briefly sketch the required changes.

3.1 Single Server Environments

Guaranteeing a consistent view on the synchronization and recovery process is straightforward in a single server environment. If a n-way mirrored volume, consisting of n

storage subsystems, is synchronized in a single server environment, the following simple changes have to be done during the synchronization or recovery process for standard accesses to the mirrored volume:

1. Write-Access: Write to each of the n storage subsystems.
2. Read-Access: Always read from the storage subsystem that acts as original volume for the building or recovery of the n-way mirror.

Running concurrently during synchronization or recovery, the original data has to be copied from the original volume to the $n-1$ mirror volumes.

Based on standard buffer caching mechanisms implemented inside the Linux kernel, no locking is required during the the synchronization process. Furthermore, a write-back of the buffer caches to the physical volumes has only to be triggered at the end of the synchronization or recovery process.

Keeping this mechanisms in mind, the kernel buffer mechanism inside Linux 2.4 and Linux 2.6 ensure a consistent synchronization of the environment.

3.2 Distributed Locking Protocols

Moving to a distributed environment, one (or more) server has (have) to take responsibility for synchronization and recovery processes. In this section, we will concentrate on synchronization. Measurement results for synchronization and recovery will be presented in section 4.

The main task in managing multi server environments is to guarantee a consistent view on the managed data. This consistency has to be ensured on two different levels:

- File System / Database: The file systems / database has to ensure cache consistency between different server working on the same file system.
- Block Level: The block level drivers do not contain enough information to guarantee a consistent view on the files and the cache usage of multiple servers. Nevertheless, the block level driver has to ensure a consistent view on the devices during synchronization and recovery of failed disks.

Without implementing a distributed locking mechanism, the synchronization and recovery of mirrored volumes will produce data inconsistencies (nearly) each time when a server accesses or changes a data block during the synchronization of this block. Therefore, the synchronization and recovery process implemented inside md has to be replaced by a distributed mechanism, implementing locking schemes for all servers.

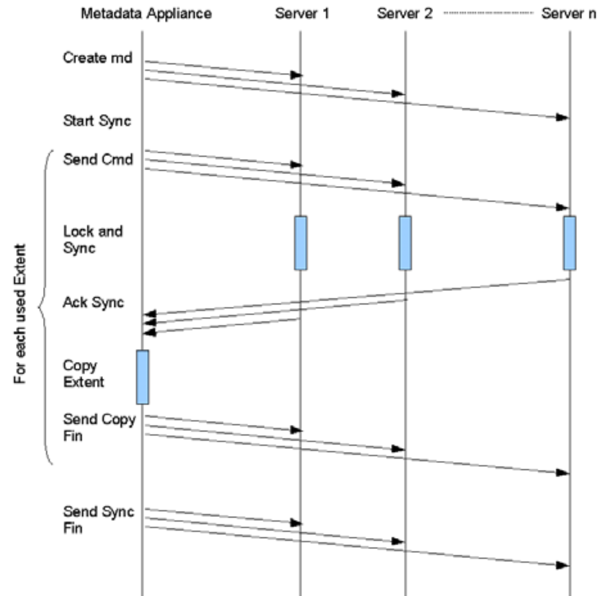


Figure 2: Distributed Synchronization Protocol.

The protocol for synchronization of an already existing virtual volume A with a new virtual volume B to form a mirror is outlined in Fig. 2.

The protocol starts by creating a new mirror on all servers assigned to the mirror. This is done by triggering a script on each server by sending a corresponding request from the Metadata appliance to each server. The md driver is used to create mirrored volumes on top of virtual volumes which have been created by the V:DRIVE environment before. To ensure consistency inside this distributed environment, the synchronization mechanism of the md driver is turned off and handled by the metadata appliance.

To disable the synchronization of disks of the original md, the creation for an md-device *mdx* has to be started with the following parameters:

```
mkraid --dangerous-no-resync /dev/mdx
```

Due to restrictions of the md-driver, it is necessary to unmount all virtual devices being part of the mirror before starting the synchronization.

After sending a create requests to each server, the metadata appliance can immediately start to synchronize the mirror without waiting for acknowledgements from the hosts, because V:DRIVE ensures a consistent view even if a server does not use the md (or at least discovers that the md volume is not used).

The synchronization is based on the concept of extents. In each synchronization round, a single extent is copied

from the original physical volume to the mirror physical volumes. The size of the extent has got an influence on the time required for the synchronization, but also on the system performance during synchronization.

Before starting the copy process, the access to the extent has to be locked in each server and information belonging to the extent has to be written back from the buffer cache of the server to the original volume (for a detailed description of this process, see the next section). After the extent has been copied to its mirror locations, the access to the extent is unlocked again. The synchronization ends after every extent has been copied to all of its mirror locations. The metadata appliance sends a synchronization finished packet to all servers belonging to the mirror and each server can switch back to normal access mode for the mirrored volume, including that it is allowed again to read from the original copy and all of its mirrors.

The time required for the synchronization process and the performance of the environment during synchronization strongly depends on the size of the extents:

- The number of synchronization rounds is equal to the number of extents. Therefore, minimizing the extent number by choosing large extents also minimizes the number of synchronization rounds.
- The probability of blocking accesses during a synchronization round increases proportional to the size of an extent (this is of course only true if accesses are equally distributed about the address space).

Measurements concerning the trade of between a fast synchronization and a high performance during synchronization are presented inside section 4.

3.3 Locking of Extents inside V:DRIVE

Besides a kernel module, each V:DRIVE server runs a small user space program that listens for TCP/IP configuration requests from the metadata appliance. One of the possible configuration requests is the request for locking and synching an extent.

Fig. 3 outlines the process for the first three steps of a synchronization round. After the user space part receives a request packet to lock and synchronize an extent, this information is forwarded to the kernel module via an ioctl()-command (see e.g. [5]). After getting the ioctl()-command, the kernel locks in a first step the extent inside the extent cache. This means that every future request is delayed inside the `make_request()`-function until the extent is unlocked again.

After locking the extent, all data belonging to the extent residing inside the buffer cache is flushed to its physical disk and the ioctl()-command returns back to user space, sending an acknowledge to the metadata appliance.

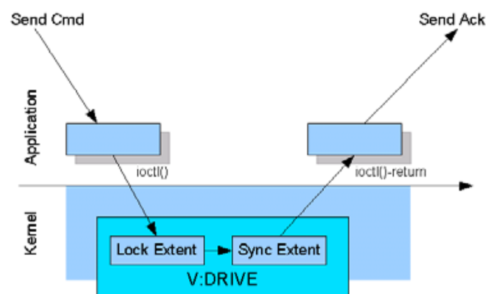


Figure 3: Locking and synchronizing an Extent.

Unlocking the extent is also done via a TCP/IP configuration request to the user space port for all servers belonging to a mirrored volume.

3.4 Special Properties of the V:DRIVE environment

In this section we will briefly discuss special properties of V:DRIVE related to the synchronization and recovery of distributed RAID environments.

As discussed in section 2, the metadata appliance of V:DRIVE keeps track on all extents inside the storage environment and it is aware on which extents have already been accessed by a simple lookup into the underlying database. This knowledge enables V:DRIVE to significantly decrease synchronization and recovery time. In V:DRIVE it is not necessary to synchronize all data that could be possibly stored on the virtual volume, but only the amount of data that has really been accessed and altered since the creation of the virtual volume. E.g., if a virtual volume with a virtual capacity of 2 TByte has to be synchronized at creation time of a mirror, but the volume only stores 200 GByte of data, only these 200 GByte of data have to be copied to their mirror locations. In this simple example, the synchronization time decreases by a factor of 10.

3.5 Transferring the concept into the original MD driver

The protocols described inside this section can be directly transferred into the original md environment, assuming that synchronization and recovery is transferred into an additional user space program, taking over the part of the metadata appliance inside the V:DRIVE environment.

Also it is simple to transfer other RAID schemes into the distributed RAID environment. For standard accesses, the md driver takes care of parity creation, for synchronization and recovery these algorithms have to be transferred into the metadata appliance.

4 Measurements

In this section, we present experimental results for the distributed md environment. The measurements focus on the performance during synchronization and recovery. All of the measurements are based on synthetic benchmarks to outline the main properties of the distributed md environment. In particular, we focus on measuring throughput and average response time of the system during synchronization and recovery.

In this section, we only present one comparison between the original md and the distributed md version. There are two reasons for this:

- During synchronization, performance is obviously better in the original md-environment. This is quite simple to see: There has not to be a write back of the cache content to the physical volumes during synchronization and parts of the synchronization can even be done inside the cache.
- After synchronization and without recovering from disk failures, there is nearly no difference in performance between the original md environment and the distributed environment inside each server. Accesses from the md driver are passed through to V:DRIVE that is also passing the accesses through to the underlying virtualized physical disks.

The only distinction between standard md and the distributed version in normal mode is that the distributed version can be configured to forward all read accesses to the primary copy of a mirror. This is helpful in many cased when mirroring is used to build a copy from a fast storage system to slower storage media.

Due to space limitations, we will not present sequential and random access results for read and write tests for all possible scenarios. Nevertheless, we have tried to mix the test cases in a way that the influence of read and write accesses and the influence of sequential and random accesses can be reconstructed from the set of presented measurements.

4.1 Measurement Environment

The performance measurements have been done using three Linux servers. Server A is a single-processor servers with a 2.4 GHz Intel Xeon processor, Server B is a double processor server with two 2.4 GHz Intel Xeon processors. Both servers have 1 GByte of main memory, a local 40 GByte hard disk, a QLogic 2310 FC HBA, and Fast Ethernet connections. Furthermore, an additional server is used as metadata appliance for the V:DRIVE environment. The metadata server is equivalent to server A. All servers run a Linux operating system with a 2.4.21 kernel. The servers are connected with each other via a Fast Ethernet switch.

All servers are connected to a Transtec 3000 JBOD system via a 2 GBit/s fibre channel network. The fibre channel array houses fourteen fibre channel hard disks with a raw capacity of 73 GByte each. Each disk is exported as two 12 GByte partitions to the servers. The disks are not grouped by any internal RAID scheme. The tests have been run using the IOMeter benchmarking environment [11]. The load generators run on Server A and Server B, the management console for the IOMeter environment runs on an additional Microsoft Windows PC, connected to the servers via a Fast Ethernet switch.

4.2 Mirror Synchronization

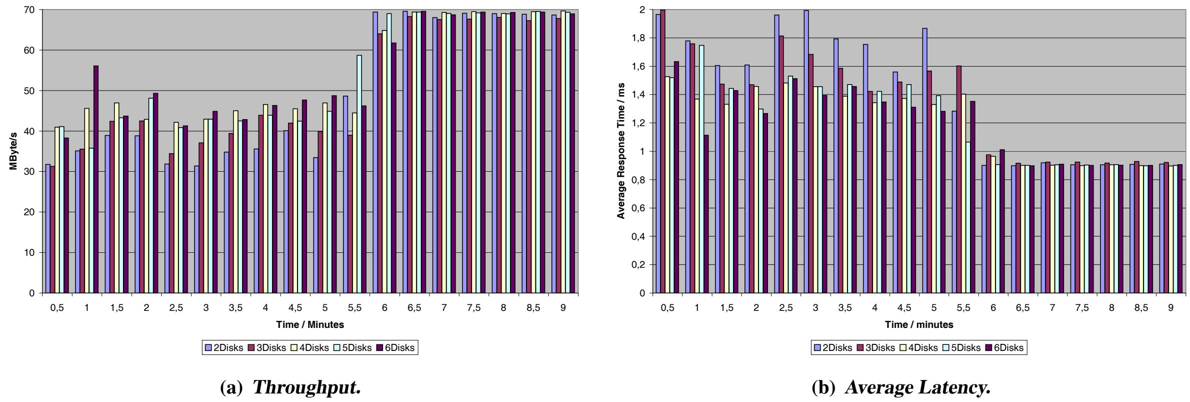
Mirror volumes can be created by either creating a mirror from scratch or by creating a mirror from an already existing volume and synchronizing the content of this volume to its mirror volume. As described in section 3, both the original md and the distributed md are able to perform synchronization on-line. This is only restricted by the obvious fact that the original virtual volume has to be taken off-line before creating (not synchronizing) and mounting the md volume.

Inside this first tests presented in this subsection, the sequential read behavior during the creation and synchronization of a distributed mirror from an already existing virtual volume is described. The aim of this measurements is to show the influence of the number of physical disks inside a storage pool on performance during synchronization. For the first test, we only use a single server accessing the data.

For the tests, two virtual volumes have been created from different storage pools. The size of the virtual volumes is 12 GByte each. All physical volumes belonging to the partitions inside the first storage pool are disjoint from the physical volumes belonging to the partitions in the second storage pool. The extent size is 16 MByte. The number of physical disks inside each pool is chosen between 2 disks and 6 disks.

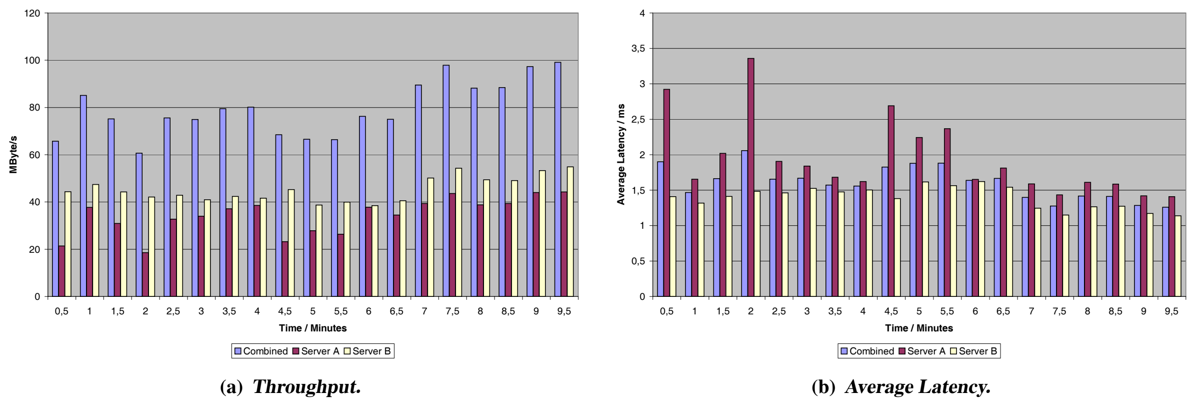
Before starting the tests, 11 GByte of data are stored on the primary virtual volume. Afterwards, the MD is created and mounted. The load is generated by a single IOMeter worker thread. The throughput and the average Latency of the system is measured reading 4 KByte Blocks sequentially. The maximum number of outstanding I/Os is 16.

Figure 4 shows the performance during synchronization dependent on the number of disks inside each storage pool. The results clearly indicate that the number of disks inside each storage pool has a big impact on system performance during synchronization. Storage throughput during synchronization is in the order of 30 MByte/s for two physical disks inside each storage pool up to 48



(a) ***Throughput.*** (b) ***Average Latency.***

Figure 4: Sequential read performance during mirror synchronization.



(a) ***Throughput.*** (b) ***Average Latency.***

Figure 5: Synchronization inside distributed environment with two servers.

MByte for 6 physical disks inside each pool (see 4(a)).

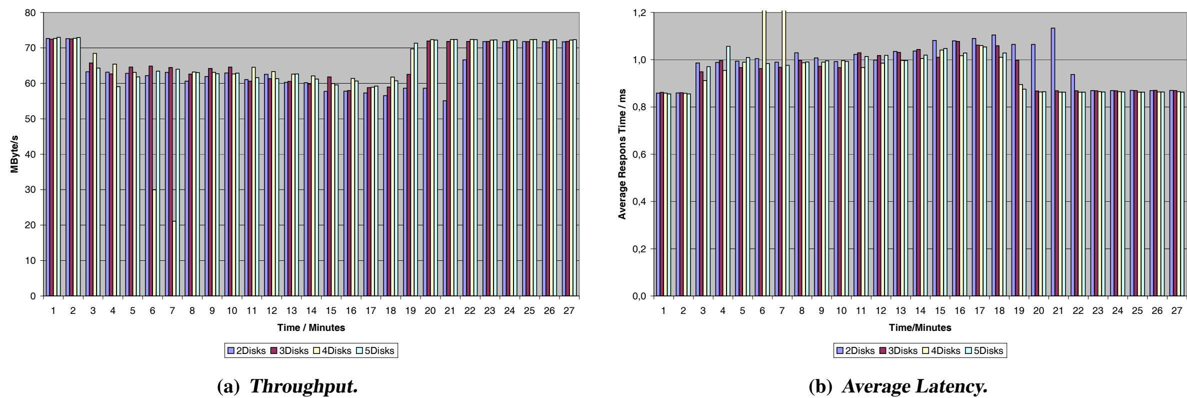
The reason for this is beside better performance for standard accesses that the distributed locking protocol synchronizes one extent at a time. Only accesses to the extent currently being synchronized are delayed and only the caches for these physical disks have to be written back, all other accesses are performed normally.

Fig. 5 outlines the synchronization behavior for two servers accessing the synchronized volumes in parallel. The tests has been performed with the same parameters as the test described before, only the size of each storage pool has been fixed to four disks. The first server accessing the mirrored volumes is a single-processor server, the second server is a dual-processor server. It can be seen that the computing performance of the server itself has also strong impact on its storage performance. It can also be seen that the combined performance of the servers is scaling well even during synchronization.

4.3 Recovery of failed Disks

One of the core features of V:DRIVE is to be able to optimally adapt to any changes of the underlying storage configuration by using an efficient replacement mechanism which copies all data from the disks that are to be removed to a new location inside the storage pool.

This subsection shows the influence of the replacement process on system performance during recovery of a failed disks. As in the section before, we created two virtual volumes with 12 GByte each from two different and disjoint storage pools, each of which initially contains 6 physical partitions. The extent size is 16 MByte. An 12 GByte md volume has been created on top of the virtual volumes and 11 GByte of data has been stored on the md volume. The measurements have been performed by a single IOMeter thread for sequential read accesses, 4KByte block size and a maximum of 16 outstanding I/Os.



(a) ***Throughput.***

(b) ***Average Latency.***

Figure 6: Sequential read performance during data recovery.

Fig. 6 shows the results for removing 2, 3, 4 or 5 disks from the system three minutes after the start of the tests. The environment then automatically starts the recovery process on the remaining disks inside the storage pool. The recovery process has taken less than 20 minutes independent on the number of failed disk while the throughput remained at nearly 60 MByte/s and the average latency has been still around 1 ms.

4.4 Influence of Extent Size on Synchronization Time

A free optimization parameter for synchronization and recovery is the extent size chosen for the storage pools. This extent size is equal to the number of rounds performed inside the distributed synchronization and recovery protocol and it is proportional to the time required to copy the extent from its original location to its target location.

Fig. 7 outlines the influence of the extent size on the performance during synchronization of a 12 GByte virtual volume. It can be clearly seen that smaller extent sizes have much smaller influence on performance during synchronization. This is especially true for the average latency, which has been up to 1 s for 512 MByte extents (see Fig. 7(b)).

It is important to notice that the tests have not started at the same point in time, even if this seems to appear from the measurement results. The time required to perform synchronization becomes smaller for larger extent sizes.

4.5 Influence of Parallelism inside MD

At the beginning of this section it has been stated that the influence of the distributed RAID enhancements can be neglected after synchronization and recovery. Accesses during normal runtime are just forwarded from the md driver through the virtual volumes under the management of V:DRIVE to the physical disks.

For the case of read accesses this is not always true. It is possible to choose inside the V:DRIVE environment, whether all accesses should be served by the primary copy or if the read requests should be distributed according to the load balancing algorithms inside md. This choice has been introduced for storage environments combined from very fast primary storage and slower archive disks.

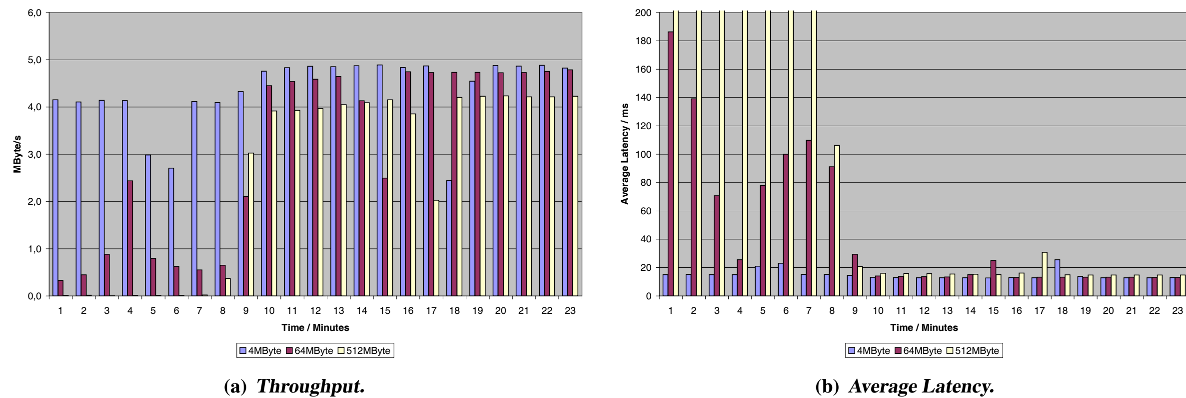
Fig. 8 outlines the influence of this choice, if all storage devices belong to the same class of storage. For these tests, random read performance for mirrored storage pools with two, respectively four disks in each storage pool has been compared. It can be seen that the performance of a mirrored storage pool with load balancing enabled is nearly as fast as the performance of a storage pool without load balancing enabled, but with double the number of disks. Therefore, the load balancing schemes inside md seem to be nearly optimal.

References

[1] A. Brinkmann, M. Heidebuer, F. M. auf der Heide, U. Rckert, K. Salzwedel, and M. Vodisek. V:Drive - Costs and Benefits of an Out-of-Band Storage Virtualization System. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pages 153 – 157, College Park, Maryland, USA, April 2004.

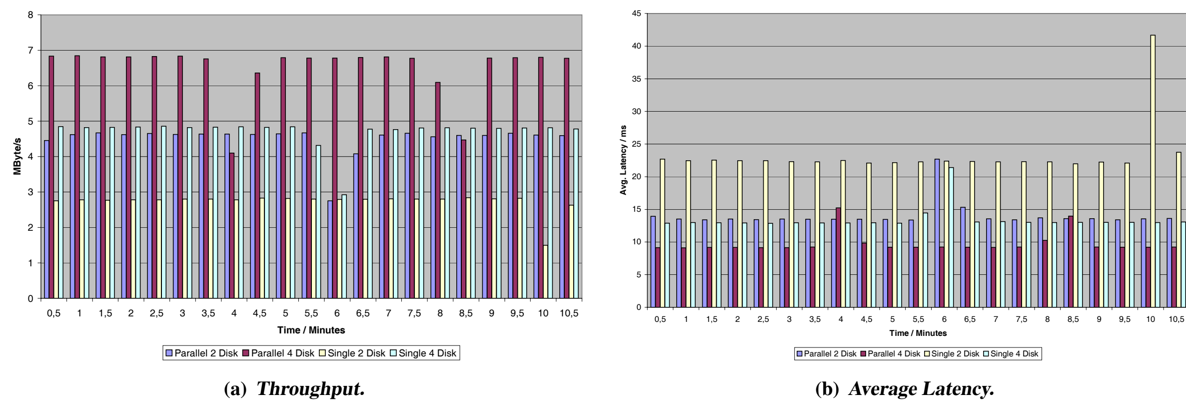
[2] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform distribution requirements. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 53–62, Aug. 2002.

[3] A. Brown and D. A. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems.



(a) ***Throughput.*** **(b)** ***Average Latency.***

Figure 7: Influence of Extent Size on Synchronization Performance.



(a) ***Throughput.*** **(b)** ***Average Latency.***

Figure 8: Performance during synchronization with and without toggled read flag.

In *Proceedings of USENIX Annual 2000 Technical Conference*, San Diego, California, USA, June 2000.

[4] P. Cao, S. B. Lin, S. Venkataraman, and J. Wilkes. The TickerTAIP parallel RAID architecture. *ACM Transactions on Computer Systems*, 12(3):236–269, 1994.

[5] J. Corbet, A. Rubini, and G. Kroah-Hartmann. *LINUX Device Drivers*. O'Reilly, 3rd edition, 2005.

[6] T. Cortes and J. Labarta. Extending heterogeneity to raid level 5. In *Proceedings of the USENIX Annual Technical Conference, Boston, MA*, pages 119 – 132, June 2001.

[7] K. Hwang, H. Jin, and R. S. Ho. Orthogonal Striping and Mirroring in Distributed RAID for I/O-Centric Cluster Computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):26–44, 2002.

[8] J. Kim, S.-W. Eom, H. N. S, and Y.-H. Won. Striping and buffer caching for software RAID file systems in workstation clusters. In *International Conference on Distributed Computing Systems*, pages 544–551, 1999.

[9] Y. S. T. Maruyama and N. Yoshizawa. Software RAID Technology for Cluster Environments. In *Proceedings of the 18th IEEE Symposium on Mass Storage Systems*. IEEE, April 2001.

[10] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.

[11] SourceForge. Iometer User's Guide. Technical report, Intel, 2003.

[12] C. Staelin. Disk I/O in Linux. Technical Report HPL-2002-352, HP Laboratories Israel, December 2002.

[13] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 90–106. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[14] Q. Xin, E. Miller, D. Long, S. Brandt, T. Schwarz, and W. Litwin. Reliability mechanisms for very large storage systems. In *20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 146–156. IEEE Computer Society, Apr. 2003.