V:Drive - Costs and Benefits of an Out-of-Band Storage Virtualization System *

André Brinkmann, Michael Heidebuer, Friedhelm Meyer auf der Heide,
Ulrich Rückert, Kay Salzwedel, and Mario Vodisek
Paderborn University

Abstract

The advances in network technology and the growth of the Internet together with upcoming new applications like peer-to-peer (P2P) networks have led to an exponential growth of the stored data volume. The key to manage this data explosion seems to be the consolidation of storage systems inside storage area networks (SANs) and the use of a storage virtualization solution that is able to abstract from the underlying physical storage system.

In this paper we present the first measurements on an out-of-band storage virtualization system and investigate its performance and scalability compared to a plain SAN. We show in general that a carefully designed out-of-band solution has only a very minor impact on the CPU usage in the connected servers and that the metadata management can be efficiently handled. Furthermore we show that the use of an adaptive data placement scheme in our virtualization solution V:Drive can significantly enhance the throughput of the storage systems, especially in environments with random access schemes.

1. Introduction

The advances in networking technology and the growth of the Internet have enabled and accelerated the emergence of new storage consuming applications like peer-to-peer (P2P) networking, video-on-demand, and data warehousing. The resulting exponential growth of the stored data volume requires a new storage architecture, while the management of the traditional, distributed *direct attached storage* (DAS) architecture has shown to be intractable from a business perspective. The first step towards this new storage architecture is the consolidation of the servers and storage devices inside a *storage area network* (SAN). In a SAN, the formerly fixed connections between storage and servers are broken-up and both are attached to the high-speed dedicated storage network. The introduction of a storage area network can significantly improve the reliability, availability, manageability, and performance of servers and storage systems.

Nevertheless, it has been shown that the potential of a SAN can only be fully exploited with the assistance of storage management software, and here particulary with the help of a virtualization system. Storage virtualization is often seen as the key technology in the area of storage management. But what actually is storage virtualization? A good definition has been given by the Storage Networking Industry Association SNIA [8]:

"[Storage virtualization is] an abstraction of storage that separates the host view [from the] storage system implementation."

This abstraction includes the physical location of a data block as well as the path from the host to the storage subsystem through the SAN. Therefore, it is not necessary that the administrator of a SAN is aware of the distribution of data elements among the connected storage systems. Generally, the administrator only creates a virtual volume and assigns it to a pool of physical volumes, where each physical volume can be of different size. Then, a file system or a database can work upon this virtual volume and the virtualization software provides a consistent allocation of data elements on the storage systems. It is even possible that a large number of virtual volumes share a common storage pool.

The use of a virtualization environment has many advantages compared to the traditional approach of assigning an address space to a fixed partition. The most obvious one is that a virtual disk can become much larger than the size of a single disk or even than a single RAID-system [7]. When using virtualization software, the size of a virtual disk is only limited by the restrictions inherent to the operating system and the total amount of available disk capacity.

Another important feature of virtualization software is a much better utilization of disk capacity. It has been shown

*Partially supported by the DFG Transferbereich 40 and the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

that in the traditional storage model only 50% of the available disk space is used. The disk utilization can be increased up to 80% through the central and more flexible administration of virtualization software. Thus, the required storage capacity and, with it, the hardware costs of a storage area network can be reduced significantly. Furthermore, virtualization software offers new degrees of flexibility. Storage systems can be added to or removed from storage pools without downtime, thus enabling a fast adaptation to new requirements. These storage systems do not have to be from a single vendor, so that the traditional vendor-locking of customers can be avoided.

Virtualization software can be implemented as out-of-band virtualization or in-band virtualization, inside the storage subsystems, or as logical volume manager (LVM) inside the hosts. In an out-of-band virtualization system, the virtualization is done inside the kernel of the hosts and all participating hosts are coordinated by one or more additional SAN appliance. In this paper we will focus on the analysis of our out-of-band solution *V:Drive*.

Chapter 2 of this paper introduces the design of *V:Drive*. In chapter 3 we present the first measurements on an out-of-band storage virtualization system and investigate its performance and scalability compared to a plain SAN. We show that a carefully designed out-of-band solution has only a very minor impact on the CPU usage in the connected hosts and that the metadata management can be efficiently implemented. Furthermore we give evidence that the use of an adaptive data placement scheme can significantly enhance the throughput of storage systems, especially in environments with random access schemes.

2. *V:Drive* Design

In this chapter we will describe the design of our out-of-band virtualization solution *V:Drive*. From the architectural perspective, *V:Drive* consists of a number of cooperating components: one or more SAN appliances which are responsible for the metadata management and the coordination of the hosts (see section 2.2), the virtualization engine inside the kernel of the hosts (see section 2.3), and a graphical user interface (GUI).

From a logical point of view, *V:Drive* offers the ability to cluster the connected storage devices into storage pools that can be combined according to their age, speed, or protection against failures. Each storage pool has its own storage management policy describing individual aspects like logical and physical block size or redundancy constraints. A large number of virtual volumes can share the capacity of a single storage pool.

The capacity of each disk in a storage pool is partitioned into minimum sized units of contiguous data blocks, so called *extents*. The extent size need not be constant inside a storage pool and can change over time. In general, smaller extents can guarantee better load balancing, while bigger extents result in a smaller management overhead and less disk head movements in case of sequential accesses. The extents are distributed among the storage devices according to the *Share* strategy which is able to guarantee an almost optimal distribution of the data blocks across all participating disks in a storage pool (see Section 2.1).

2.1. The Share-Strategy

Any virtualization strategy depends on the underlying data distribution strategy. Such a distribution is challenging if the system is allowed to contain heterogeneous storage components. The main task of a distribution strategy is an even distribution of data blocks and an even distribution of requests among the storage devices. Therefore, it has a strong impact on the scalability and the performance of the SAN. It can be shown, that a static data placement scheme is generally not able to fulfill the given requirements.

We have developed a new adaptive distribution scheme that has been implemented in *V:Drive*, called *Share*-strategy [2]. In this paper we will present *Share* without data replication. Of course it is possible to support replication inside *Share*, e.g. by a scheme proposed in [4]. For other static and dynamic placement schemes, see [6, 3, 4, 5].

Share works in two phases. In the first phase, the algorithm reduces the problem of mapping extents to heterogeneous disks to a number of homogeneous ones. The result is a number of volumes which are equally likely to store the requested extent. In the second phase, we use any distribution strategy that is able to map extents to equal sized disks (see e.g. [6]).

The reduction phase is based on two hash functions $h : \{1, ..., M\} \to [0, 1)$ and $g : \{1, ..., N\} \to [0, 1)$ where M is the maximal number of extents in the system and N is the maximal number of disks that are allowed to participate, respectively.

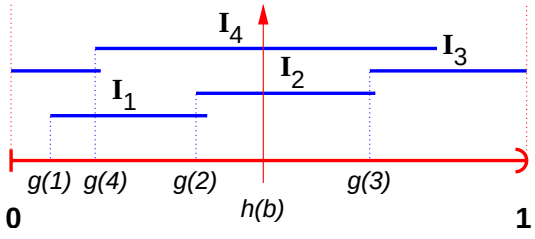

Figure 1. *Hashing scheme in Share*

The reduction phase works as follows: Initially or after every change in the system configuration, we map the

starting points of sub-intervals of certain length into a [0,1) interval using the hash function *g*. The length of these sub-intervals I_i corresponds to the capacity d_i of disk *i*. To ensure that the whole interval is covered by at least one sub-interval we need to stretch each of the sub-intervals by a factor *s*. In other words, the sub-interval I_i starts at $g(i)$ and ends at $(g(i) + s \cdot d_i) \bmod 1$.

The extents are hashed into the same interval using *h* where the quality of *h* ensures an even distribution of all extents over the whole interval. Now, an extent can be accessed by calculating its hash value and then deriving all sub-intervals that value falls into. Any efficient uniform strategy can be applied to get the correct disk out of the number of possible candidates. It can be shown that the fraction of extents stored on a disk and the number of requests to a disk are proportional to its capacity and that the number of extent replacements in case of any change in the number or kind of disks is nearly minimal (see [2] for more detail).

2.2. SAN Appliance and Metadata Management

To ensure a consistent view of the SAN and a proper configuration of the hosts and storage devices, one or more SAN appliances are connected to the SAN. The appliances keep track of all necessary metadata structures. These metadata include among others the partitioning of the storage devices into storage pools and virtual volumes, access rights of the hosts, and the allocation of extents on the storage devices.

The metadata appliance consists of a number of separate modules which are arranged around the *V:Drive* database, including the *Disk-Agent*, the *Host-Interface*, the *Disk-Manager*, and the *Administration Interface*. The interface to the database is standard SQL, implemented in many commercial and free databases. All components of the appliance can be executed on a single machine or can run in a distributed fashion.

Information about the state of the SAN are collected by the *Disk-Agent* that is responsible for detecting disk partitions and for finding changes in the system configuration. Each newly detected suitable partition is labelled with a unique ID and is made available by updating the database.

The *Host-Interface* is connected to the servers via Ethernet/IP. A data transfer between a server and the host interface is issued if the configuration of the SAN has been changed, if the server has started and has to load its configuration, or if the server accesses a virtual address for the first time and has to allocate an extent on the corresponding disk.

If the configuration of the storage system changes, a small number of extents has to be redistributed in order to guarantee close to optimal performance. The *Disk-Manager* is responsible for this redistribution tasks. After each change of a storage pool it checks each allocated extent if it has to be relocated. In such a case, the extent is moved online to its new location in a way that ensures the consistency of the data before, during, and after the replacement process.

The administrator can access the metadata via the graphical user interface. The administration interface contains all the necessary functionality to manage enterprise wide storage networks: administration of storage systems, storage pools, and virtual devices, authentication and authorization, security, and statistics.

2.3. Kernel Integration

The host software basically consists of a kernel module which is linked to the operating system of the participating servers and some additional applications running in the user space. Currently, modules for the Linux kernel 2.2 and 2.4 are available.

If a data block needs to be read from a virtual disk, the file system generates a block I/O request and passes it to the kernel module where it is processed and transmitted to the appropriate physical disk. To perform the transformation from a virtual address to a physical address, the kernel keeps all necessary information, like existing storage pools, assignments of virtual and physical disks to the pools, storage policies etc. These information are given to the kernel initially or on-demand by the metadata server.

3. Results

In this section we will present the experimental results of our virtualization approach. The test system consists of two Pentium servers connected to an FC-AL array with 8 fibre channel disks. Both servers have 2 Pentium II processors with 450 MHz and 512 kilobyte cache. Furthermore, they have local access to a mirrored disk drive containing the operating system and all relevant management information. Both servers run a Linux 2.4.18 kernel (Red Hat) and use gcc version 3.2.2 as the C compiler. The access to the disks is enabled by a QLogic qla2300 host bus adapter. The FC-AL array consists of four 17 Gigabyte and four 35 Gigabyte fibre channel disks. They are connected with the server via an 1 Gigabit switch. Each disk is partitioned into one partition covering the whole disk.

For stressing the underlying I/O subsystem we used the Bonnie file system benchmark [1]. We changed the original source code such that we could derive more information concerning the overhead of our solution. The simple design and easy handling of Bonnie makes it a suitable tool for testing I/O performance. It performs, among others, the following operations on a number of file of desired size: it

reads and writes the random content of each character of the file separately, it reads and writes each block of the file, and it concurrently accesses arbitrary blocks in the file.

The first two tests access a number of data files sequentially. Such a scenario is rather unlikely in practice but it is able to give an idea of the maximal performance of the I/O subsystem. More suited to model real world scenarios is the last test, because we have to access arbitrary blocks in some files. We set the overall file size to 4 GB (4 times the size of the main memory) to reduce caching effects. The size of the extents was fixed to 1 MB.

To derive the overhead of our approach we compare our approach to the performance of a plain disk (labeled with the device name, e.g. SDA). More specific, we investigate the influence of each component of our solution to the overall throughput. For that we distinguish the following cases:

1. Clean System (C): Nothing is known in advance.
2. Transfer (T): All extents exist in the database and have only to be transferred to the driver.
3. Driver (D): The driver has all information locally and does only perform the mapping of addresses.

The number in parentheses behind the letters C,T,D in the charts axes is equivalent to the number of physical volumes belonging to the corresponding storage pool. If not mentioned otherwise, the storage pool consists of a single physical volume.

Throughout the experiments the CPU usage for our approach was indistinguishable from the CPU usage when accessing the plain disk. Due to space limitations the corresponding figures are omitted.

3.1. Impact of Extent Requests

Figure 2 shows the throughput for the different settings when each character is written separately. Note, that we only get an overhead when the extent is accessed for the first time. Therefore, the induced costs are credited to many data accesses and their effect becomes marginal. The differences are mostly due to cache effects.

The situation is very different when it comes to blockwise accesses. Here, the fraction of block accesses which induce overhead is much higher. Figure 3 shows the performance not only for the different settings but also for varying sizes of corresponding storage pools. Surprisingly, we lose roughly 40% when using only one disk. The reason for that lies mostly in the special sequential access pattern. The achieved high throughput could only be gained because the layout of the data blocks on disk enables a sweep of the disk head, minimizing the head movements. Modern file systems take that into account and adapt their data layout accordingly. But we destroy the careful layout because we

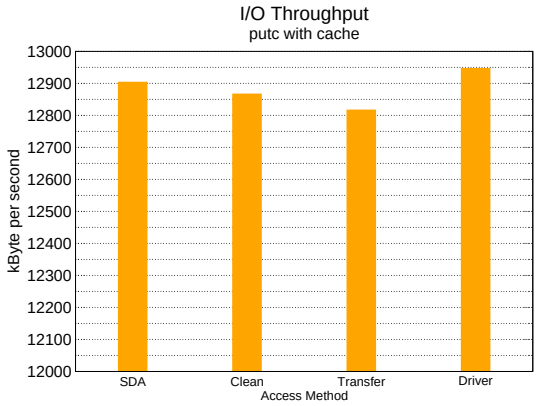

Figure 2. Comparison of the sequential output per character.

access the data in extents instead of data blocks. When allocating an extent the metadata server returns the first free position on the disk that is big enough to host the extent. Therefore, a sequential access of the file system results in higher movement of the disk head and only the sustained throughput of a disk could be achieved.

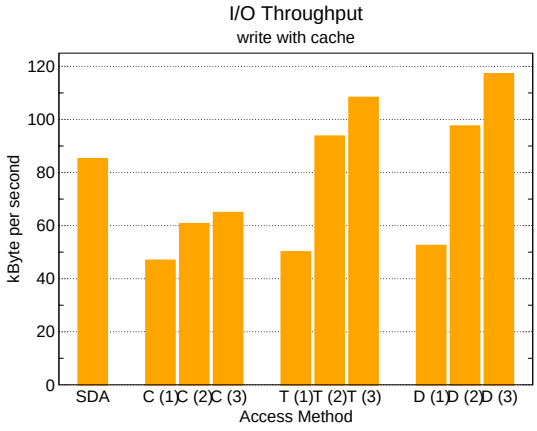

Figure 3. Comparison of the sequential output per block.

Surprisingly, this effect is compensated by parallel accesses when feeding the virtual device from more than one disk. Due to the fact that the operating system issues the write requests to the main memory and returns immediately, we achieve parallelism and get roughly the sustained performance of two disks. We could top the performance significantly, even if the access pattern does not allow for much parallelism. This indicates that the overhead induced by the driver alone is not a limiting factor. Only a clean sys-

tem with many extent allocations is not able to use many disks to increase the performance compared to a single disk. But in a real-world application a data block is accessed many times and the overhead occurs only once.

3.2. Block Read Performance

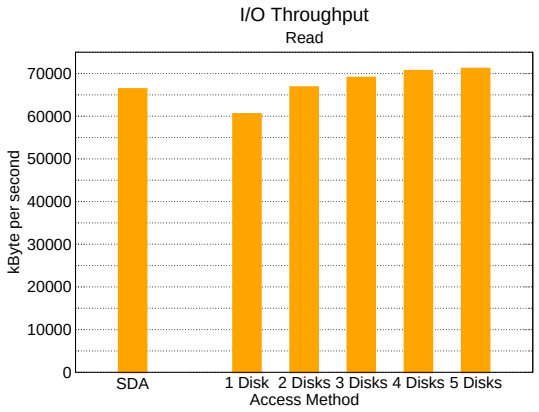

Figure 4. Comparison of performance of block accesses.

The read access is different from a write because it has to wait until the data is delivered from the disk. This gives the operating system enough time to rearrange a larger number of data accesses and, hence, accesses the disk in a sweeping manner. Figure 4 gives evidence for that. We lose only about 9% compared to the performance of a plain disk. As noted above the access pattern allows little parallelism. Hence, the increasing of the number of disks has only a small impact on the overall performance.

3.3. Random Seeks

To get the number of random seeks per second Bonnie creates 3 threads performing the data requests. It is our opinion that this test is closest to practice because on a storage server there are different application generating rather unpredictable block accesses. Figure 5 compares the number of seeks per second for all approaches. Again, the overhead induced by the *V:Drive* solution is too small to measure once the extents are allocated.

Note, that the impact of more disks decreases the more disk participate in the storage pool. This is due to the fact that the number of scheduled requests stays constant. That means, that the likelihood of parallel accesses to all disks decreases with the number of disks. If we would access the storage pool with more virtual devices the scaling would be much better.

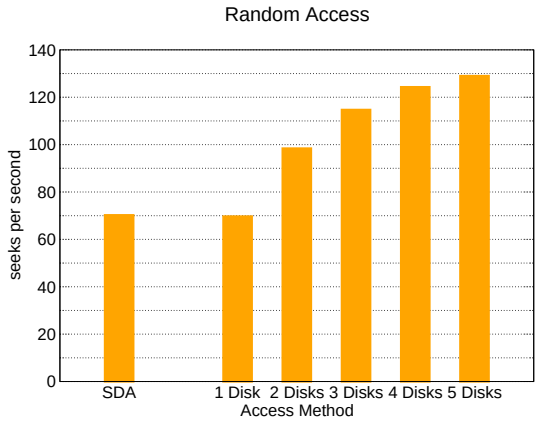

Figure 5. Comparison of the performed number of seeks per second.

4. Conclusion

In this paper we presented a virtualization environment that is based on the randomized Share-strategy. The shown results give evidence that such an approach is not only feasible but also efficient. Especially the performance of random seeks to files via Bonnie hints that *V:Drive* scales nicely with a growing storage network.

References

[1] T. Bray. Bonnie source code. http://www.textuality.com.

[2] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, Adaptive Placement Schemes for Non-Uniform Distribution Requirements. In *Proceedings of the 14th ACM SPAA Conference*, 2002.

[3] T. Cortes and J. Labarta. Extending Heterogeneity to RAID level 5. In *Proceedings of the USENIX Annual Technical Conference*, 2001.

[4] R. J. Honicky and E. L. Miller. A Fast Algorithm for Online Placement and Reorganization of Replicated Data. In *Proceedings of the 17th IPDPS Conference*, 2003.

[5] R. J. Honicky and E. L. Miller. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In *Proceedings of the 18th IPDPS Conference*, 2004.

[6] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *In Proceeding of the 29th ACM STOC Conference*, pages 654–663, 1997.

[7] 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)*, 1988.

[8] The Storage Networking Industry Association (SNIA). Storage Virtualization I: What, Why, Where and How.