Towards Workload-Driven Adaptation of Data Organization in Heterogeneous Storage Systems

Nikolaus Jeremic[1], Helge Parzyjegla[1], Gero Mühl[1], and Jan Richling[2]

[1] Architecture of Application Systems Group,
University of Rostock, Germany
{nikolaus.jeremic,helge.parzyjegla,gero.muehl}@uni-rostock.de
[2] Communication and Operating Systems Group,
TU Berlin, Germany
richling@cs.tu-berlin.de

Abstract. Collecting, managing, and analyzing huge data sets (i.e., Big Data) in science and industry poses challenges to current data storage systems in terms of storage capacity, performance, and reliability. In particular, the I/O performance may be a key factor to speed up the data analysis. However, the performance of a storage system significantly depends on its configuration and the access pattern. Designing storage systems always implies making compromises between performance, fault tolerance and net capacity. The decision which compromise is made (e.g., which RAID level is used) has to be taken at deployment time because runtime reconfigurations are usually prohibitively expensive (due to coarse granularity) in current storage architectures.

In this paper, we propose a workload-driven approach to adaptive reconfiguration covering the functionality of the file system, volume manager and RAID. Our approach enables fine-grained reconfigurations of the data organization of files and file fragments to adapt the storage system to changing workloads, while considering the different characteristics of the storage devices (e.g., SSDs and HDDs) in a heterogeneous storage system. We first discuss how our approach decreases the costs of adaptations compared to existing approaches making a continuous and effective adaptation feasible, even for large volumes of data. Then, we present an evaluation based on a prototypical implementation confirming the benefits of our approach.

1 Introduction

The capability to manage, process, and analyze data sets of extreme size, diversity, and complexity is becoming a key requirement in modern business. It enables companies and organizations to mine such sets of Big Data for valuable information in order to faster gain enhanced insight into trends, customers, and markets as well as to better identify opportunities and assess the risks. In this context, storage systems are an essential prerequisite that links the collection of data with its subsequent analysis. First and foremost, a storage system has to provide enough capacity to accommodate Big Data volumes. But beyond that,

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 33–42, 2014.

the speed with which data can be accessed is particularly important [6] in order to accelerate the processing and analysis of data sets. With the advent of new storage technologies such as flash memory as built into Solid-State Drives (SSDs), more storage alternatives to conventional Hard Disk Drives (HDDs) become available that, on the one hand, promise a significant performance gain and speed up, but on the other hand, currently offer only limited capacities at much higher costs. To profit from both worlds, heterogeneous storage systems provide an elegant solution.

Heterogeneous storage systems often comprise several drive pools each featuring a different technology [2,4] and/or data organization, e.g., RAID scheme [11]. Please note that besides the storage technology, data organization also has a major impact on *performance* (with respect to bandwidth and latency of requests), *reliability* (with respect to drive failures), and *usable storage capacity* [5]. Each data item is, then, stored in the pool which best meets its performance, reliability, and capacity requirements. Moreover, if the access pattern changes, the data item can be migrated to a different pool that better suits the new pattern which renders the storage system adaptive. In general, this is leveraged to maximize the performance for those data items accessed most frequently[1].

However, the coarse granularity of adaptations provided by current storage systems severely limits the effectiveness of this approach. In fact, only logical volumes are often subject to migration although access statistics within the volume may vary widely. Thus, decisions are based on mean values and also affect those portions of the volume that actually do not profit from migration. Even if sub-volume data units are considered [4], as done by more advanced storage systems, these units still have the size of several megabytes. This alleviates the issue, but does not solve the problem. Similarly, adapting the size of the storage pools or changing their RAID policy is usually avoided [11] due to the coarse granularity of the required data reorganization and associated reorganization costs. For instance, removing a drive from one pool and adding it to another with a different RAID configuration, requires a complete data reorganization on the block layer for both pools. In particular for Big Data volumes, it is nearly impossible to accurately estimate whether this huge reorganization effort will pay off in reasonable time.

In this paper, we present a workload-driven approach for an automated fine-grained reconfiguration of storage systems. By adapting the storage configuration in tiny steps, we can capture access patterns much more accurately in large volumes while minimizing the migration and reorganization overhead at the same time. For this purpose, we apply RAID policies at sub-file level and track access patterns at this granularity in order to decide on reorganizations based on the evaluation of benefits and costs. We provide a case study and evaluation that proves the feasibility of the approach and demonstrates its advantages.

[1] Alternatively, caching is often used to speed up access times for frequently used data. Please note, that caching is an orthogonal concept that can be used in addition to data migration and RAID policy adaptation, respectively.

The remainder of this paper is structured as follows: Sect. 2 introduces the background on RAID systems. Based on that, we present our approach in Sect. 3. Sect. 4 describes our case study, followed by the experimental evaluation. Finally, the paper is concluded in Sect. 5.

2 RAID

RAID (Redundant Array of Independent Disks) [9,8] denotes a technique for combining a drive collection into a single logical drive to increase the performance, reliability and storage capacity. This is achieved through data *striping* (declustering), *mirroring* and/or adding *parity*. A *RAID policy* is defined by means of different attributes and their values: the *RAID level*, the number of used drives and configuration parameters specific to the RAID level (e.g., the stripe unit or chunk size, the number of data copies). The RAID level determines the data organization and which of the concepts above are used. Each RAID policy provides a different trade-off between performance, fault tolerance, and usable storage capacity. Thus, an improvement regarding to one of the characteristics often goes along with a degradation in at least one of the others.

A RAID policy can be specified and implemented on different levels within the storage (software) stack, which has major implications for the policy's scope and the reorganization overhead in case of a policy change. Most RAID systems are implemented on the block layer and, thus, allow to specify a policy on a per-volume basis. More advanced implementations additionally subdivide volumes in smaller units in the range of several megabytes and allow to set a RAID policy for a group of such units. Changing the RAID policy requires the reorganization of the complete volume or, in the case of sub-volume units, considerable parts of it causing substantial reorganization overhead. Since on the block level, it cannot be distinguished whether a block contains user data or is actually unused, the overhead is always proportional to the size of the reorganized storage space.

More recently, RAID logic has been integrated into file systems such as *Btrfs* or *ZFS*. Nevertheless, the RAID policy applies to whole volumes, and is eventually implemented on the basis of blocks. Reorganization support is still rudimentary or even missing although it may be implemented in a clever way considering only used blocks. However, when RAID and file system implementations are combined, there is no reason to not allow a more fine-grained specification of the RAID policy, e.g., on the basis of directories or even individual files as proposed by Appuswamy et al. [1]. This increases flexibility but does not always reduce the overhead of potential data reorganizations in case of policy changes because large files may have the size of or may even be bigger than small volumes.

3 Automated RAID Policy Adaptation

As motivated in Sect. 1, automated reconfiguration of the RAID policy is a promising mean to address changes in access pattern. However, reconfiguring large volumes containing assets of Big Data suffers from high costs due to the

coarse-grained application of common RAID policies. Therefore, we propose to apply reconfigurations as fine-grained as possible based on the trade-off between cost and benefit that is derived from continuous monitoring of access pattern and requirements (specified by application or user).

The idea is to decompose files into small segments and specify the RAID policy for each segment that can later be individually changed and adapted. The data chunks of each segment are then distributed over available drives according to the segment's RAID policy. Disjoint drives are randomly selected when a new segment is created in order to spread larger requests over multiple drives and balance the load. The exact mapping is stored in metadata and is independent of the chunk's position within the segment. Depending on the modified attributes, this can avoid relocations of the chunks when the RAID policy is changed. Obviously, this approach needs a tight integration into the file system. In this paper, however, we neglect integration details and solely focus on an automated adaptation. In particular, this includes the general concepts to derive an adaptation policy that takes application requirements and reorganization costs into account.

3.1 RAID Reconfiguration Control

The control of RAID reconfigurations is based on the evaluation of system parameters and settings as well as on optimization objectives and priorities between them, thus, basically restricting possible reconfigurations. If, for instance, a particular level of fault tolerance needs to be ensured for certain files, a number of more efficient, but less reliable RAID policies may not be applicable. For the remaining policies, optimization objectives such as performance or storage capacity help weighting the gains of different alternatives. The selection of the RAID policy is based on a generic bonus-malus system which also triggers the execution of corresponding RAID reconfigurations.

To optimize performance, for instance, the data layout of a file segment is adapted to the most frequent access pattern. Thus, if the current layout is well suited for a particular request, a bonus is granted encouraging the current RAID policy, otherwise a malus is taken into account. Bonus and malus are set off against each other, whereby the bonus-malus ratio is maintained for each file segment as score. The score is increased when a malus is taken into account and decreased in the case of bonus. However, the score is reset to 0 when it has dropped below 0 in order to keep the heuristic reactive to pattern changes. If the score reaches a predefined upper bound, the RAID policy is changed and a corresponding reconfiguration is triggered. Hence, when defining the upper bound, a threshold depending on the reorganization overhead needs to be considered. To reduce the latency of the request that triggered the reconfiguration, the reconfiguration can take place after the request has been served.

If optimizing for storage capacity, space-efficient data layouts periodically get a bonus, while others receive a malus. In the meantime, however, the space-inefficient layouts may also score bonus points if data accesses are served performantly. As consequence, primarily cold data gets reorganized first, while performance-oriented RAID policies are kept for hot data as long as possible.

With respect to reliability, we assume that a minimum level of fault tolerance is defined for each data item. Therefore, reconfigurations due to optimization of performance or storage capacity consider only RAID policies meeting this level. If the defined reliability level is low enough, performance optimization may result in striping without redundancy, trading reliability for performance and capacity.

3.2 RAID Reconfiguration Costs

A data layout reorganization in the course of RAID reconfiguration causes additional read and write requests to the drives that dominate the reconfiguration overhead. Hence, we discuss the overhead in terms of such requests. The reconfiguration overhead depends on which changes of the RAID policy are performed. In all cases described below, the number of read requests can be reduced if the affected data is located in the page cache. Changing the RAID level corresponds to switching between striping (either with or without parity) or mirroring. In the event of switching from striping to mirroring, each chunk of a stripe has to be read and replicated multiple times in order to produce the desired number of copies. However, the original chunk can be reused saving one write. Switching from mirroring to striping requires to read the mirrored data and write it to several stripe units. Additionally, in the case of striping with parity, it is required to calculate and write the parities.

Changing the number of drives used in a RAID policy also has a major impact. In the case of mirroring, this translates to increasing or decreasing the number of copies. Adding copies requires reading a chunk once and writing it to each additional drive. Decreasing the number of copies, however, may not require accessing the data at all. In the case of striping, the change of stripe width necessitates to relocate at least a subset of chunks in each stripe of which each must be read and written. If parity is involved, all parities have also to be recalculated and rewritten. Moreover, assigning chunks to drives independently of their position within the stripe can avoid relocations in certain situations, e.g., when parity is added/removed and the stripe width is correspondingly adjusted.

4 Case Study: RAID Level Reconfiguration

This section describes an exemplary instance of the approach proposed in Sect. 3. The objective is to maintain high write throughput by adapting the RAID level based on the request size under the constraint that no data should be lost in the case of up to two arbitrary drives failing simultaneously. Therefore, we consider RAID 10 with triple mirroring and RAID 6 and fix the number of drives to the minimum of six. Furthermore, we also fix and use the same maximum chunk size for both RAID policies. Thus, in case of reconfiguration either a file segment in RAID 6 layout is split into two RAID 10 segments or two adjacent RAID 10 segments are merged into one RAID 6 segment. Both reconfiguration cases do not require a relocation of data chunks, reducing the reorganization overhead.

Table 1. RAID 6 and RAID 10 layout scores for HDDs and SSDs

Affected Chunks	RAID 6 HDDs	RAID 10 HDDs	RAID 6 SSDs	RAID 10 SSDs
1	56	29	30	10
2	55	53	28	20
3	56	72	24	38
4	54	94	21	39

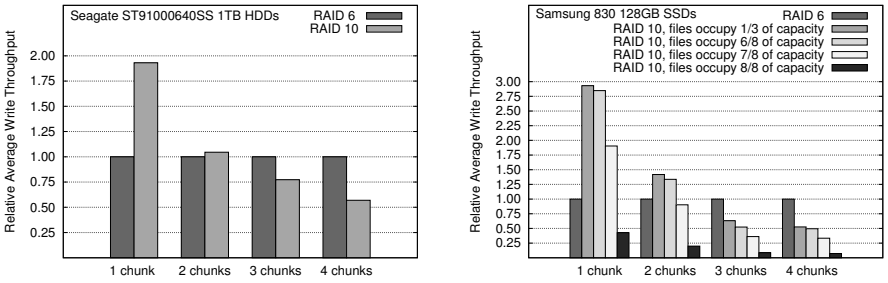
4.1 Reconfiguration Policy

The write throughput depends on the number of affected chunks within a file segment which results from the request size. The reason is that write requests to a file segment translate to different number of subsequent chunk reads and writes depending on the number of affected chunks. The assessment of each RAID layout, with respect to the number of affected chunks and device type (HDD or SSD), is based on measured average throughput (the measurements are described in Sect. 4.2). The considered write overhead is determined by averaged write latency, which is translated into scores summarized in Table 1, where 1 point equals 0.1 ms for HDDs and 0.01 ms for SSDs. For example, according to the first line in Table 1, a write affecting one chunk takes 5.6 ms on HDDs for RAID 6, but only 2.9 ms for RAID 10, saving 2.7 ms when performed in the better suited RAID 10 layout. The considered bonus or malus equals the difference in write latency between the RAID 6 and RAID 10 layout with respect to the device type.

After completing a write request to a segment, its RAID level is switched in the case that its score exceeds a threshold. A reasonable minimum value for the threshold is given by the reconfiguration overhead, but it should be set higher in order to avoid oscillations between the two configurations. The considered reconfiguration overhead is based on measurements and corresponds to the average reconfiguration time (the corresponding measurements are described in Sect. 4.2). Switching from RAID 6 to RAID 10 requires 9.9 ms on HDDs and 0.54 ms on SSDs. The reconfiguration threshold is set to 100 for HDDs and to 55 for SSDs. Switching from RAID 10 to RAID 6 requires 6.5 ms on HDDs (which lies below the average access time of 8.5 ms [10] due to seek time optimizations performed by the HDDs when multiple requests are submitted in parallel) and 0.29 ms on SSDs. Thus, the threshold is set to 66 for HDDs and to 30 for SSDs.

4.2 Experimental Evaluation

A server-class machine (2x Intel Xeon E5-2680, 128 GiB RAM, LSISAS2308 controllers) equipped with Seagate ST91000640SS HDDs and Samsung 830 SSDs served as testbed. On the software side, Linux (kernel 3.7) was used and all measurements were obtained using *fio* 2.0.13 [3]. The workload consisted of synthetically generated file I/O traces. The RAID reconfiguration (explained in Sect. 4.1) was emulated by translating file I/O traces into resulting I/O requests



(a) HDDs (b) SSDs on varying space utilization

Fig. 1. Average write throughput for RAID 10 compared to RAID 6 considering different number of affected chunks (64 KiB per chunk)

to block devices before measurement. Then, these requests were submitted to drives using *fio* and employing direct, native Linux asynchronous I/O. Furthermore, the *noop* I/O scheduler was used and *NCQ* was enabled for all drives. To minimize the impact of caching, the on-drive caching was disabled on HDDs, however, not on SSDs because read caching cannot be disabled using *hdparm* or *sdparm*. In all experiments, the maximum chunk size was 64 KiB. Each experiment was repeated three times and the mean value was taken as result. In the case of SSDs, the *TRIM command* was applied between runs to discard all data.

The assessment of the considered RAID configurations described in Sect. 4.1 is based on measurements of the average write throughput on varying number of affected chunks, which were performed separately for HDDs and SSDs to consider the different device type characteristics. Write requests were carried out on a set of files on randomly chosen file segments with request sizes based on the intended number of affected chunks: 64 KiB for 1 chunk, 128 KiB for 2 chunks etc. Moreover, to consider the influence of spare capacity on the write performance of SSDs [7], file sets of different sizes were used (Fig. 1(b)). For each RAID layout and device type, the average throughput was measured for writing an amount of data corresponding to 60 GiB.

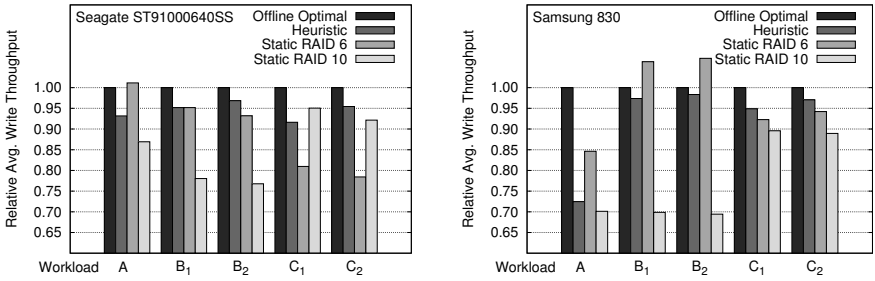
The measurements for HDDs depicted in Fig. 1(a) show that RAID 10 is more suitable for writes affecting up to 2 chunks, while RAID 6 should be preferred when 3 or 4 chunks are affected due to the higher throughput. The results for SSDs in Fig. 1(b) indicate the same trend, while the differences between RAID 10 and RAID 6 are more pronounced for writes affecting up to 2 chunks. However, the measurements also show that RAID 10 throughput is highly sensitive to the amount of occupied capacity and, hence, of available spare capacity. SSDs perform out-of-place updates, thus, if our data set occupies as much capacity that further write requests will trigger the garbage collection, the write performance drops significantly (cf. [7]). This effect is emphasized in cases where the file set occupies the whole drive capacity, which is represented by the black bars in Fig. 1(b). However, RAID 6 is less sensitive to this effect since its data layout occupies only half of the storage capacity (in our setup) compared to RAID 10.

Therefore, the throughput remains roughly the same. Please note, that we cope with the sensitivity of SSDs to the amount of spare capacity in the remaining experiments by choosing the amount of written data such that the storage capacity of each SSD is not completely utilized. The reconfiguration thresholds mentioned in Sect. 4.1 are based on measurements of the average reconfiguration time on HDDs and SSDs. For both types of reconfiguration and each device type 320,000 file segments (80 GB file data) were reconfigured in random order. The overall duration was measured and used to calculate the average reconfiguration time.

The proposed heuristic for adaptation of the RAID level to the write request size was compared to an offline algorithm as well as to static RAID 6 and RAID 10 setups. An offline algorithm determines which initial RAID level and reconfigurations lead to the highest write throughput with respect to the empirical cost model described in Sect. 4.1 based on the a priori knowledge of all write requests to each file segment. For the sake of comparing the RAID level adaptation to static setups, the average write throughput was measured based on five synthetically generated workloads (denoted as A, B_1, B_2, C_1, and C_2) with different characteristics that are described in the following. Each workload represents overwriting data in randomly chosen file segments (in a set of files comprising 20 GB data) with different request sizes, which leads to different number of affected chunks. The overall amount of data written to a set of files equals 200 GB for each workload. In workload A, the size of each write request is chosen at random, hence, each number of affected chunks is almost equally likely. Unlike this, in the remaining workloads B_1, B_2, C_1, and C_2, a particular file segment receives a number of recurring writes that last for a certain period, whereby this period is four times longer for B_2 and C_2 than for B_1 and C_1, respectively. Moreover, the request size distribution of B_1 and B_2 is biased to larger requests affecting 3 and 4 chunks. In contrast to this, C_1 and C_2 are dominated by smaller requests, i.e., 1 chunk and 2 chunks.

The results for HDDs depicted in Fig. 2(a) indicate that the RAID level adaptation heuristic leads to a notable performance gain, if a certain write pattern persists for a longer amount of time, which applies to the workload B_2 and C_2. For both workloads, the average write throughput reaches over 95% of throughput achieved by the offline algorithm, whose optimization decisions are based on the a priori knowledge of all performed write requests. However, if the write pattern changes more frequently (as in workload B_1 and C_1), the heuristic can trigger reconfigurations that will not pay off, thus, even leading to lower performance than a static configuration. This applies for workload C_1, where the heuristic reaches a notably lower throughput than RAID 10. However, this advantage seems less when the write pattern changes, e.g., making the RAID 6 setup the better choice. Beside that, the results for workload A indicate a minimal deviation from the empirical cost model, due to approx. 1% higher write throughput in the case of RAID 6 compared to the offline algorithm.

The results for SSDs depicted in Fig. 2(b) show that the heuristic outperforms both static setups for workloads dominated by small requests (C_1 and C_2).



(a) HDDs (b) SSDs

Fig. 2. Relative average write throughput for HDDs and SSDs compared to offline model-optimal RAID level adaptation

However, as C_1 and C_2 are dominated by small requests, RAID 10 is expected to be the better choice according to the write overhead results shown in Fig. 1(b), where RAID 10 clearly outperforms RAID 6, which seems to clash with the results for C_1 and C_2 (where RAID 6 slightly outperforms RAID 10). This stems from the recurring writes to a file segment that increase the temporal locality of reference, which is exploited by the on-drive caching. Please note, that RAID 6 also incurs subsequent chunk reads in the case that writing a file segment affects up to 3 chunks, which does not apply to RAID 10. Thus, when all chunk reads are served from cache the difference between RAID 6 and RAID 10 is evened out if a write affects 1 or 2 chunks. Furthermore, read caching performed by SSDs also widens the performance gap between RAID 6 and RAID 10 for workloads B_1 and B_2, and additionally introduces a remarkable deviation from the empirical cost model (RAID 6 performs better than the offline algorithm). However, for workload A the impact of caching is less pronounced due to lower temporal locality.

5 Conclusion

The performance of secondary storage systems is crucial, in particular, when processing Big Data. It may be the decisive factor to accelerate and speed up the data analysis allowing it to complete within a given time frame. In this paper, we tackled the challenge of adapting the data organization in heterogeneous storage systems to access patterns and storage requirements (e.g., performance, reliability, and capacity). We have proposed a scheme for a workload-driven RAID policy adaptation that was employed in order to increase the performance. To avoid prohibitive reorganization costs due to moving potentially large amount of data, our approach is based on RAID policies that are applied at sub-file level. This permits reacting to fine-grained changes of the workload. We evaluated this approach using a case study and an initial implementation, showing both, its applicability and its advantages. However, our results also make clear that

the characteristics of different device types and caching have to be properly considered in order to avoid that optimization turns into the opposite.

As very large data sets have often to be distributed over several storage servers, we plan to continue our research in two directions: First, we will further improve the adaptation strategies and consider the current requirements on storage space and reliability as well as extending the reconfiguration to further parameters. This requires to refine the model of the secondary storage system by taking device-specific performance properties and the impact of caching into account. In addition to this, it may also be beneficial to consider hints provided and requirements posed by applications. Second, we will extend our strategies to scenarios, where data sets are distributed over multiple storage servers. In this case, global information on access patterns, demands, and system state need to be gathered and exploited to improve the overall data organization.

References

1. Appuswamy, R., van Moolenbroek, D.C., Tanenbaum, A.S.: Block-level RAID is dead. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Storage and File Systems, HotStorage 2010, pp. 1–5. USENIX Association, Berkeley (2010)
2. Appuswamy, R., van Moolenbroek, D.C., Tanenbaum, A.S.: Integrating flash-based SSDs into the storage stack. In: Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST 2012), pp. 1–12. IEEE (2012)
3. Axboe, J.: Flexible I/O tester (2013), http://freecode.com/projects/fio
4. Guerra, J., Pucha, H., Glider, J., Belluomini, W., Rangaswami, R.: Cost effective storage using extent based dynamic tiering. In: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST 2011, pp. 1–14. USENIX Association, Berkeley (2011)
5. Jacob, B.L., Ng, S.W., Wang, D.T.: Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann (2008)
6. Jacobs, A.: The pathologies of big data. Commun. ACM 52(8), 36–44 (2009), http://doi.acm.org/10.1145/1536616.1536632
7. Jeremic, N., Mühl, G., Busse, A., Richling, J.: The pitfalls of deploying solid-state drive RAIDs. In: Proceedings of the 4th Annual International Conference on Systems and Storage (SYSTOR 2011), pp. 14:1–14:13. ACM (June 2011), http://doi.acm.org/10.1145/1987816.1987835
8. Katz, R., Gibson, G., Patterson, D.A.: Disk system architectures for high performance computing. Proceedings of the IEEE 77(12), 1842–1858 (1989)
9. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (RAID). SIGMOD Rec. 17, 109–116 (1988)
10. Seagate Technology: Product manual Constellation. 2 SAS. Tech. rep., Seagate Technology (April 2012), http://www.seagate.com/files/www-content/support-content/documentation/product-manuals/en-us/Enterprise/Constellation_2_5in/100620418h.pdf
11. Wilkes, J., Golding, R., Staelin, C., Sullivan, T.: The HP AutoRAID hierarchical storage system. ACM Trans. Comput. Syst. 14(1), 108–136 (1996), http://doi.acm.org/10.1145/225535.225539