Improving RAID-5 performance by reducing data redistribution time

Genti Daci
Polytechnic University of Tirana
Faculty of Information Technology
Sheshi Nënë Tereza 4, Tirana, Albania
gdaci@acm.org

Mirela Ndreu
Polytechnic University of Tirana
Faculty of Information Technology
Sheshi Nënë Tereza 4, Tirana, Albania
mirela.ndreu@fti.edu.al

ABSTRACT
RAID is a storing technology that is very often used in nowadays systems and in general this storing volume provides more data security, availability and performance to the existing system. Being an important component of the system, the data storage volume and its performance affect directly the performance of the entire system. This is the motivation why it is convenient for us to improve as much as it is possible the weaknesses of the RAID volume. We are focused on data redistribution problem which is necessary to be performed after adding disks to the volume. In this paper we will review some algorithms and in particular we will discuss FastScale approach, which resulted in satisfactory values of data redistribution time when it was applied on RAID-0. Our proposal is a short algorithm that interacts with FastScale and adjusts it for RAID-5 volumes. The main structure of FastScale is not changed and this gives us reason to anticipate lower redistribution time and higher performance in RAID-5 volumes, as happens in RAID-0.

Categories and Subject Descriptors
D.4.3 [**Operating Systems**]: File System Management – *maintenance;* D.4.3 [**Operating Systems**]: Performance – *modeling and prediction.*

General Terms
Algorithms, Performance.

Keywords
RAID-5, data redistribution time, parity block.

1. INTRODUCTION
RAID systems give higher performance, more capacity and data reliability to the existing system. RAID volumes can be managed as a single device and they are useful for the support of every kind of user application [1]. They are very often used in nowadays systems to support data storage. As part of the different components that are present in systems, the performance of the entire system depends on RAID performance also. Efforts to increase even more the performance of RAID 5 led us to the weakness of this device. One situation that we are always faced with is a continuous increase of user data and thus the continuous requirement for larger storage capacity. To supply this necessity we perform a disk addition. This disk addition is termed *"RAID scaling"*. The problem that is to be solved after scaling is the redistribution of data onto all the disks of the volume.
As hardware and software of systems have changed and improved, also different techniques have developed for redistribution of data in different types of systems [6], [7], [8], [9]. We will make a general summary of some of the data redistributing approaches. Especially, we will discuss the features of every technique used by them. Initially we will take a look on SCADDAR approach. It is an efficient, online method to scale disks in a continuous media server and it is based on using a series of REMAP functions which derive the location of a new block using only its original location as a basis [5]. SCADDAR ensures load balancing of blocks and in general low complexity computation to the system. The development of scaling approaches has advanced achieving other levels of performance and efficiency.
We will review also another approach that is based on the fact that during the data redistribution process and there is always a reordering window where no valid data chunk will be overwritten while changing the order of data movements [3], [22], [23]. The reordering window is a window where data consistency can be maintained while changing the order of chunk movements and its characteristic provides a theoretical basis for solving the problem of scaling RR-striped volumes. SLAS approach utilizes the features of the reordering window and guarantees data consistency and does not enlarge the impact on the response time of foreground I/Os. Experiments have shown that SLAS has a good performance, but during redistribution it moves all the blocks of data into all the disks and this have been proved that is a weakness for SLAS that decreases the performance. We are interested in finding an approach that performs the redistribution of data, but it also minimizes system loading and redistribution time. Further, there is ALV approach that increases the efficiency of a scaling process based on the reordering window and applies it on RAID-5 storage volume [4], [16], [17], [18]. It takes advantage of the qualities of the reordering window and uses three added techniques to make it appropriate for RAID-5. From the experiments result it was concluded that ALV had a noticeable improvement over earlier approaches in two metrics: the redistribution time and the user response time. Based on the fact that we are interested in the redistribution of data in RAID-5, ALV is an interesting case, but the development of techniques has progressed even more. FastScale is an algorithm that best fulfills our requirements for a performing and efficient approach [2], [15], [20], [21]. It is implemented and proven in RAID-0 and the experiments values are the reason why we decided to treat this algorithm and to make it adaptable for RAID-5. The significant results of this approach derive by its addressing function. FastScale has a very small redistribution time against SLAS, because its addressing function minimizes data movement.. In Figure 1 is showed the movement of data blocks on RAID-0 with FastScale. FastScale is not implemented in RAID-5 because it has not included parity bit in its addressing function During the review of FastScale we noticed that the blocks change move from

old disks to the added ones, but they do not change the physical address. Our contribution in this paper is exactly the modification of FastScale. We add some lines to include the parity bits on FastScale. The position of the parity block is important in RAID-5 volumes and we take account of them in our algorithm.

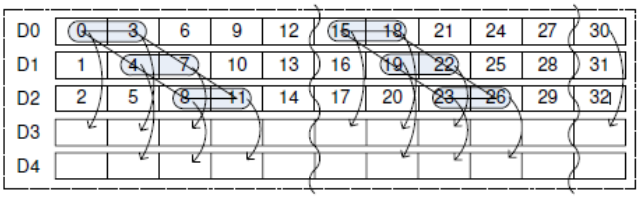

Figure 1. The moving of blocks in RAID-0 after adding two new disks using FastScale.

We rely on the fact that the main strengths of FastScale that improve the performance of RAID-0 are not changed, and thus we anticipate that even in RAID-5 will have short redistribution time and high performance.

2. SCALING TECHNIQUES

2.1 Data migration techniques on different systems

2.1.1 SCADDAR

The first requirement that the approach must fulfill is making redistribution of data without interruptions of the activity. SCADDAR [5] is an approach that was developed according to this requirement for the continuous media server. This is the example of one of the approaches that performs an efficient, online scaling process when we add disks in a continuous media server.

The redistribution is oriented by pseudo-random placement without moving all the blocks after each scaling operation. SCADDAR requires a storage structure for recording scaling operations, rather than a directory for storing block locations, as it happens in other cases. The storage structure for the scaling operations in SCADDAR is significantly less than the number of all block locations. One of the particularities of SCADDAR is the computation of the new locations of blocks on-the-fly for each block access by using a series of inexpensive mod and div functions. In practice, it uses a series of REMAP functions which derive the location of a new block using only its original location as a basis. The redistribution_funciton() and access_function(), that are the main components of the SCADDAR approach, satisfies all the objectives that were presented. It makes the movement only over those blocks that have the necessity to move, and thus blocks either move onto an added disk or off of a removed disk. REMAP always uses a new source of randomness to compute the remapped number of the block. Also, block accesses only require one disk access per block.

After several experiments that was performed with SCADDAR, it resulted that it provides load balancing. In other words, SCADDAR maintains load balancing of blocks across disks after several scaling operations. After eight scaling operations performed on 20 different objects, the percentage of load fluctuation reaches the threshold level in which redistribution of all blocks is recommended. The uniform distribution, the balanced load after redistribution, the retrieved redistributed blocks at the normal mode of operation and the low complexity computation are the restrictions that SCADDAR satisfies.

2.1.2 CRUSH

CRUSH (Controlled Replication Under Scalable Hashing) is a scaling approach that supports different hierarchy levels that provide the administrator finer control over the data placement in the storage environment. It works like a pseudo-random, deterministic function that maps an input value, and in practice this value is an object or object group identifier, to a list of devices on which to store object replicas. The difference from conventional approaches in that data placement is that it does not rely on any sort of per-file or per-object directory. The only condition that should met from CRUSH before it performs the data movement is the need to have a compact, hierarchical description of the devices comprising the storage cluster and knowledge of the replica placement policy. Its main advantages are: first, it is completely distributed such that any party in a large system can independently calculate the location of any object; and second, what little metadata is required is mostly static, changing only when devices are added or removed. CRUSH is designed to optimally distribute data to utilize available resources, efficiently reorganize data when storage devices are added or removed, and enforce flexible constraints on the object replica placement that maximize data safety in the presence of coincident or correlated hardware failures. CRUSH supports different data safety mechanisms, including *n*-way replication (mirroring), RAID parity schemes or other forms of erasure coding, and hybrid approaches (*e. g.*, RAID-10). These are the real features that make CRUSH suited for managing object distribution in extremely large (multi-petabyte) storage systems where scalability, performance, and reliability are critically important.

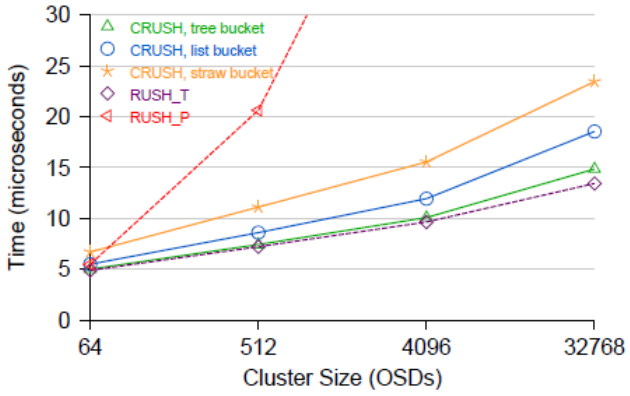

Figure 2. The computation time of CRUSH compared to RUSH.

Fig.2 illustrates the result of one of the experiments that were performed in comparison with RUSH [9]. In general, all experiments, demonstrated that CRUSH's performance—both the execution time and the quality of the results are better. In its present form, approximately 45% of the time spent in the CRUSH mapping function is spent hashing values, making the hash key to both overall speed and distribution quality and a ripe target for optimization.

2.1.3 Other techniques

The development of the scaling and data redistributing techniques includes a variety of situations according to the system where we want to make the distribution. For example, DRP is a technique that was focused on the mathematical modelling of the problem of disk replacement, when one disk fails and it was proved that the best solution is a variation of the single-source shortest path problem [6]. The upper bound on the complexity of finding the

optimal sequence is polynomial. DRP guarantees that after every atomic operation the data is balanced across the devices involved; and also after every atomic operation all the block locations are well randomized, i.e., The workload imposed on every device is approximately equal. Migrating the data and respecting a constraint on the total number of available disk slots was a condition to be ensured when a sequence of disk additions and removals of a storage system is found. There are two term the operation: if the system has enough empty disk slots to add all new disks the operation is termed *unbounded*; otherwise, it will be *bounded*. The DRP's limitation was that the authors have not addressed the limited storage of each disk. It is possible in some cases that algorithms may exceed the physically available storage on some devices temporarily.

Another approach that was thought to scale to exascale environments, as either their memory consumption, their load deviation, or their processing overhead is too high, is Random Slicing. In general, the other approaches are able to easily adapt to changing environments, a property which cannot be delivered by table- or rule-based approaches, but the Random Slicing strategy combines the advantages of different approaches by keeping a small table and thereby reducing the amount of necessary random experiments. The evaluations and comparisons with well known strategies studied by the authors [7], [10], [24] shows that Random Slicing is able to deliver the best fairness in all the cases studied and to scale up to exascale data centers. During the comparison of different hashing-based data distribution strategies that are able to replicate data in a heterogeneous and dynamic environment, there were shown the strengths and drawbacks of the different strategies as well as their constraints. Random Slicing overcomes the drawbacks of randomized data distribution strategies by incorporating lessons learned from table-based, rule-based and pseudo-randomized hashing strategies [25], [26]. It keeps a small table with information about previous storage system insertions and removals. This table helps the most to reduce the required amount of randomness in the system and thus reduces the amount of necessary main memory by orders of magnitude. One important note is that all randomized strategy map (virtual) addresses to a set of disks, but do not define the placement of the corresponding block on the disk surface. This placement of the block devices has to be resolved by additional software running on the disk itself. This way to manage the addition is different from the conventional block-based hard drives, object based storage devices that manage disk block allocation internally, exposing an interface that allows others to read and write to variably-sized, arbitrarily-named objects.

2.2 Scaling approaches on RAID systems

Our attention was focused on RAID systems, and especially on level 5 RAID. As we mentioned before, the redistributing techniques have developed time after time, improving performance of different systems where they are applied. In this subsection we have brought a short summary of two scaling approached that are developed for RAID systems.

2.2.1 SLAS

When we make a division of the striping policies, we can mention: round-robin policy and random policy. This classification is based on one of the most important problems in current systems, the increasing demand of applications for higher I/O performance and larger storage capacity. Random striping appears to be more flexible when adding new disks or deleting existing disks. But random striping is not as much a satisfactory solution as expected because its poor performance and lack of qualified randomized hash function. Round-robin striping, instead, gives to the system uniform distribution and low-complexity computation and this makes usable it the most of applications that demand high bandwidth and massive storage. The storage systems where round-robin striping is applied in are: disk arrays, logical volume managers, and file systems. We add disks to the round-robin striped volumes when storage capacity and I/O bandwidth of many systems need increasing.

The basis of the SLAS approach starts when the concept of the reordering window was defined. The need to have another data redistribution approach generated the researches on the reordering window. During the data redistribution process, there is always a reordering window where no valid data chunk will be overwritten while changing the order of data movements. The reordering window is a window where data consistency can be maintained while changing the order of chunk movements and its characteristic provides a theoretical basis for solving the problem of scaling RR-striped volumes.

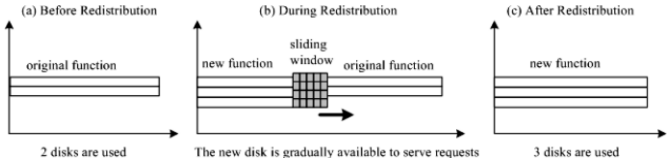

Figure 3. Mapping management based on a sliding window for the data redistribution

Fig. 3 illustrates the concept of the sliding window during the process of redistribution. The sliding window is similar to a small mapping table, and it describes the mapping information on a continuous segment of the striped volume. Before the data redistribution, the original mapping function is used, and 2 disks are used to serve requests. During the data redistribution, only data within the range of the sliding window are redistributed. The foreground I/O requests, sent to the logical address in front of the sliding window, are mapped through the original function; those sent to the address behind the sliding window are mapped through the new function; and those sent to the address in the range of the sliding window are mapped through the sliding window.

After all of the data in the sliding window are moved, the window slides ahead by one window size. Thus, the newly added disk is gradually available to serve foreground I/O requests. The data redistribution of the whole volume is completed when the sliding window reaches the end of the original striped volume. SLAS guarantees data consistency and does not enlarge the impact on the response time of foreground I/Os. SLAS changes the movement order of data chunks in a sliding window in order to aggregate reads/writes of multiple data chunks and SLAS serves foreground I/O requests between aggregate chunk reads/writes in a disk-scaling operation. The data redistribution causes the increase of the number of metadata writes. SLAS uses an additional technique to decrease this number: lazy updates of metadata mapping.

Among the SLAS features we have to mention that it can not only be used to add new disks to a RAID-0 volume; it can also be extended to remove existing disks from a RAID-0 volume and to add/remove disks to/from a RAID-4 or RAID-5 volume [3]. The experiments made with SLAS demonstrated that it shortens the redistribution duration and the maximum response time. We are interested also on another SLAS feature: during redistribution it moves all the blocks of data into all the disks and moving all data

blocks is not necessary because this reduces system performance. Later we will discuss another approach that changed in exactly this feature of the technique and its results a performing solution.

2.2.2 ALV

ALV is another approach [2] that increases the efficiency of a scaling process based on the reordering window applying it on RAID-5 storage volume. The main achievement of the authors was to take advantage of the qualities of the reordering window and then they used different techniques to make it appropriate for RAID-5. In general, every approach coordinates with other techniques that increase performance level. The three techniques that ALV uses to improve RAID-5 performance are the following: first, ALV changes the order of data movements to access multiple successive chunks via a single I/O. Second, ALV updates mapping metadata lazily to minimize the number of metadata writes. Data movement is not check pointed, until a threat to data consistency occurs. And third, depending on application workload, ALV adjust the redistribution rate using an on/off logical valve. The operation mode of ALV approach is similar to SLAS approach because of their common basic technique: the reordering window. Using the new techniques, ALV achieves higher efficiency. It was concluded from the experimental results that ALV had a noticeable improvement over earlier approaches in two metrics: the redistribution time and the user response time.

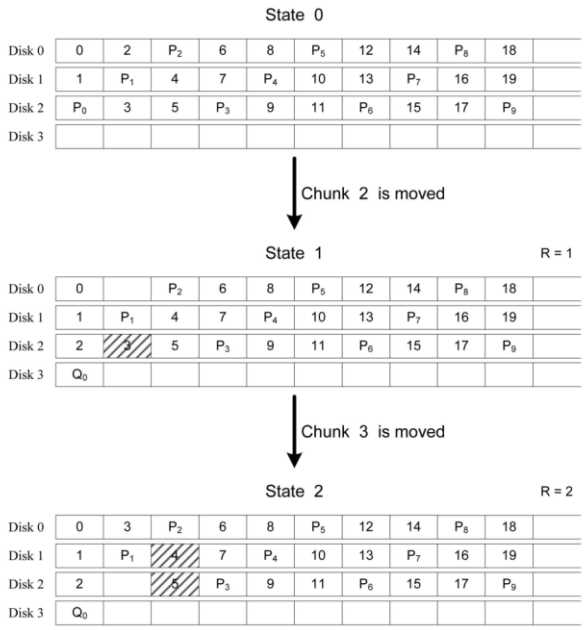

Figure 4.A series of states in data redistribution for RAID-5 scaling from 3 disks to 4. The reordering window is represented by "R".

The difference between RAID-5 and RAID-0 is precisely the presence of the parity bits, and in one way or another, these bits will influence the scaling process. In the Fig. 4 are illustrated the initial states of the redistribution process in RAID-5 volume using ALV. The presence of parity blocks orients all the blocks movement. This example of the process proves that the reordering window solves properly the influence of the parity blocks. In the figure, "P" represents the parity before scaling and with "Q" is noted the parity that will be calculated after the redistribution process. ALV changes the order of the block movement and this gives the possibility to avoid unnecessary parity blocks, and to recalculate the new parity blocks.

ALV is derived from SLAS and we found an approach that has a better performance compared to SLAS. Our focus is precisely on RAID-5 volumes and this is why we will propose an algorithm that can be applied to it.

2.3 FastScale

Before proposing our approach we will summarize the main features of FastScale. It is an approach that tolerates multiple disk addition moving the minimum amount of data. The basic idea of the FastScale approach is shown in Fig. 5.

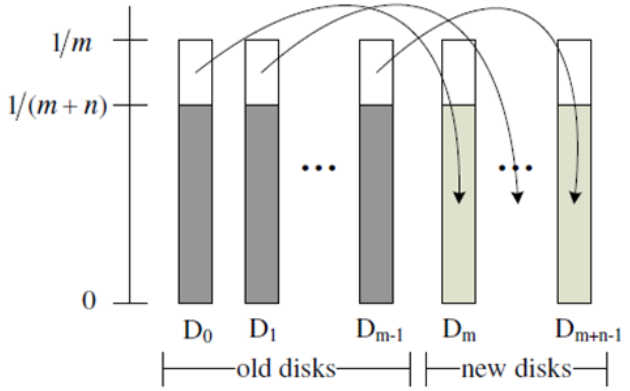

Figure 5. Data migration using FastScale. No data is migrated among old disks.

FastScale moves only data blocks from old disks to new disk enough for preserving the uniformity of data distribution, while not migrating data among old disks. The main strength of FastScale is its elastic addressing function. This addressing function computes easily the location of one block, without any lookup operation. FastScale changes only a part of the data layout while preserving the uniformity of data distribution. So, FastScale minimizes data migration for RAID scaling during the redistribution process.

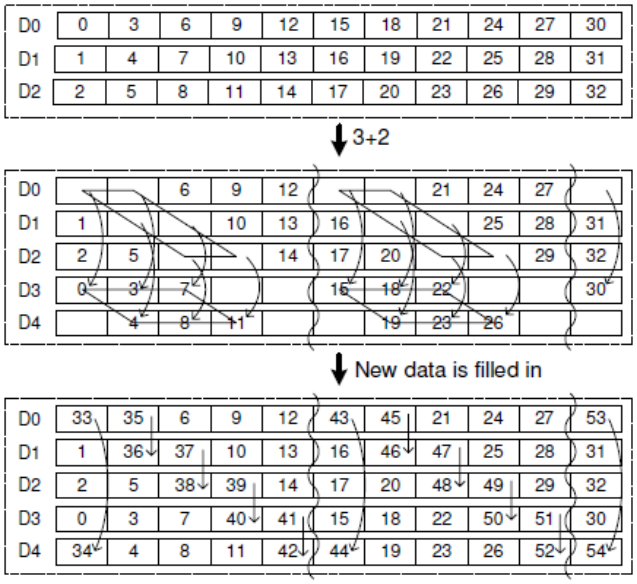

Figure 6. The stages of the scaling process in RAID-0 using FastScale.

One RAID scaling process can be divided into two logical stages: data migration and data filling. In Fig. 6 are shown both of the stages, a fraction of existing data blocks is migrated to new disks and then filled in. For the RAID scaling, we group into one segment each 5 sequential locations in one disk. For the 5 disks, 5 segments with the same physical address are grouped into one region. In the figure, different regions are separated by a wavy line. The data migration and data filling process is the same for every different region. In a region, all of the data blocks within a parallelogram will be moved. The base of the parallelogram is 2, and the height is 3. In other words, 2 data blocks are selected from each old disk and migrated to the new disks. The example in Fig. 6 shows the general operation that FastScale makes to the data blocks during the migration process in RAID-0.

FastScale satisfies all the requirements of a scaling algorithm. FastScale maintains a uniform data distribution after RAID scaling; minimizes the amount of data to be migrated entirely; preserves a simple management of data due to deterministic placement; can sustain the above three features after multiple disk additions. The success of FastScale depends also on other special physical optimization made to the process of data migration. It uses aggregate accesses to improve the efficiency of data migration. It records data migration lazily to minimize the number of metadata updates. However, data consistency is ensured, even metadata updates are minimized.

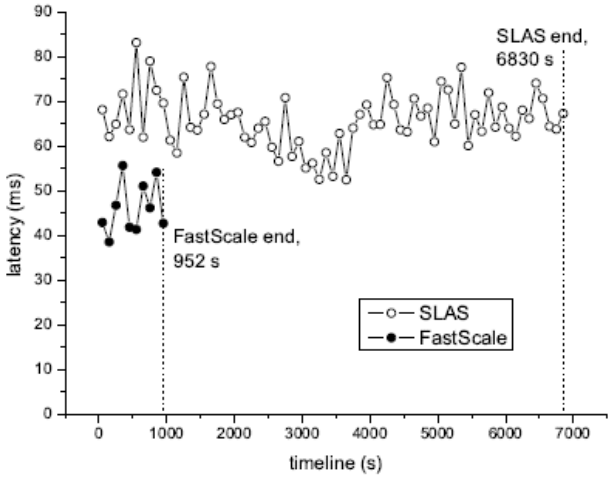

Figure 7. Performance comparison between FastScale and SLAS under the same workload.

Fig.7 shows graphically the results of a comparison made between FastScale and SLAS under the same workload. All the results of the experiments done [2] show that FastScale has a high performance even in different workload. FastScale is implemented and proved on RAID-0 volumes when we add disks, but it is not implemented when we remove disks. Otherwise for RAID-5 it is not implemented yet, because the factor of the parity bits is not taken into account in the addressing function of the approach.

3. THE PROPOSED ALGORITHM FOR RAID-5

Our goal is to give to RAID-5 scaling a higher performance using the techniques that FastScale owns. The restriction of FastScale to RAID-5 is that it does not include parity bits in the algorithm. We worked exactly on including the parity bits in the scaling process. Looking carefully the structure of RAID-5, we notice that the position of every parity block is defined by a certain rule. In RAID-5, as it happens in RAID-0, we group blocks in regions. As it is shown in Fig.8, in every region of a RAID-5 volume we have a parity block in every disk and these blocks correspond to different physical addresses. In other words, there is only a parity block for every physical address of the system. After the addition of new disks the parity blocks change their position and their value. Our algorithm gives a solution how FastScale can include the parity blocks in the redistribution process.

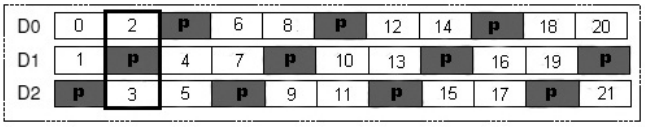

Figure 8. The structure of RAID-5 and the blocks with the same physical number.

The other fact that we will use in our algorithm is: during the redistribution, FastScale does not change the physical number inside of blocks, but they only move from old disks to the new ones with the same physical address. Basically our approach does not change anything to FastScale approach. The redistribution process after scaling RAID-5 is conducted form FastScale and the blocks move regardless of the content of the block.

Our proposal must be implemented exactly after the migration of data blocks and before the computation of the new parity value. At first, our algorithm controls if the position of the parity block is right, then it calls the parity computation procedure. The control if the position of parity is right includes three situations:

- The new position of parity block is empty
- The new position of parity block has the "old" parity block
- The new position of parity block has data written in it

The first and the second situation are less problematic, because we can write and overwrite the "new" parity safely, without losing data. The third situation requires more attention. If there is written data in the block, the new parity cannot be written there. Given to the facts that parity blocks have specific positions, and that there must be an "old" parity block, that is not more needed anymore, with the same physical number *b* and we can make an exchange between them. This way, we save the data and write the parity bits where it is required.

Our algorithm works in addition to the function that calculates parity. We have not defined a specific function for the calculation of the parity.

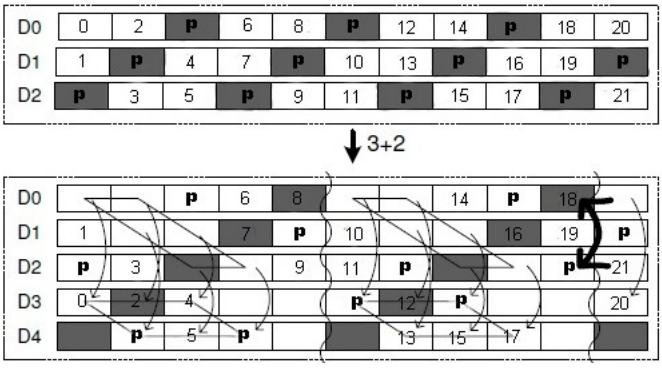

Figure 9. The redistribution process in RAID-5 using the proposed algorithm.

In Fig.9 is illustrated schematically the process of data redistribution on RAID-5 using our algorithm. Initially, this algorithm makes a control of the content of every position that should be a parity block. If the block is empty, the function of the parity calculation is called. Otherwise, if the block is not empty, we must distinguish if the content is parity or data bits. In the case of parity bits, we do the same as it was empty: we write over it because the old parity is not needed, the new one is written. If the content is data, we cannot write over it, because we lose the data. In this case, we find the old parity block that has the same physical number and perform an *exchange* of blocks. On one side, we protect data and save them, and on the other side we write the new parity bits in the proper position.

It is explained by W. Zheng and G. Zhang [2], that FastScale gives high performance by minimizing the data movement. The phase of the calculation of parity is unavoidable in RAID-5 volumes and it adds latency to the process of redistribution. But, due to the fact that we do not change any part of the addressing function of FastScale, we predict that the performance of RAID-5 with FastScale will be at high levels too.

The proposed Algorithm:

ParityBitPositionControl (*m, n, d, b*)

d: the disk holding block x
b: physical block number
m: the number of old disks
n: the number of new disks

if R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1][*b*] == null
// we control if the physical block is written
d_0 ← R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1], b_0 ← *b*
// we define d_0 and b_0 like the coordinates of the position of the new parity
ParityComputationProcedure (d_0, b_0)
// we call the procedure that calculates the new parity
exit;
else if R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1] [*b*] == parity bit
d_0 ← R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1], b_0 ← *b*
R[d_0][b_0] ← null
// the old parity is not necessary anymore, so we delete the information in it, and then we write the new parity in it
ParityComputationProcedure (d_0, b_0)
exit;
else if R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1] [*b*] == info bit
d_0 ← *R[(m+n) – b*mod *(m+n-1) - 1], b_0 ← *b*
// it is necessary to save logical block of the striped information, so when we have information in the position where the new parity should be, we move it to the physical block of the old parity
R[*m* – *b*mod (*m* - 1) – 1][*b*] ← R[d_0][b_0]
R[d_0][b_0] ← null
// after moving info bits, the old parity is not necessary, so we can delete it
ParityComputationProcedure (d_0, b_0)
exit;

R[x][y] *– is noted the position of the block in the whole system.*
ParityComputationProcedure *– this instruction calls the procedure that is used in the system to calculate parity.*

4. CONCLUSIONS

Data storing devices are an important component of nowadays systems and their performance affect to the entire system performance. RAID volumes are storing devices that provide more data security, availability and performance to the existing system. Our attention was concentrated on RAID-5 and especially in finding a way how RAID-5 could be even more efficient and to improve the performance of the system. Due to the fact that there is a continuous need to add disks to the volume, then we face with the problem of data redistribution. After reviewing some algorithms that perform the redistribution, further we discussed FastScale that is an approach with high levels of performance. The contribution of this paper is an algorithm that interacts with FastScale and includes the parity blocks in the redistribution process of data blocks. This algorithm makes possible the application of the FastScale algorithm and its techniques to RAID-5 also. We rely on the fact that the main strengths of FastScale which enhance performance on RAID-0 are not changed, and therefore we anticipate that even in RAID-5 will have short redistribution time and high performance. However, the focus during our future work will be on proving our prediction through experimental values and numerical analysis.

5. REFERENCES

[1] D. A. Patterson, G. A. Gibson, R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID), in Proceedings of the International Conference on Management of Data (SIGMOD'88), June 1988. pp. 109-116

[2] W. Zheng and G. Zhang. Fastscale: Accelerate raid scaling by minimizing data migration. In Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST), Feb. 2011.

[3] G. Zhang, J. Shu, W. Xue, and W. Zheng. SLAS: An efficient approach to scaling round-robin striped volumes. ACM Trans. Storage, volume 3, issue 1, Article 3, pg 1-39, March 2007.

[4] Guangyan Zhang, Weimin Zheng, Jiwu Shu, "ALV: A New Data Redistribution Approach to RAID-5 Scaling," *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 345-357, March 2010.

[5] A. Goel, C. Shahabi, S-YD Yao, R. Zimmermann. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), pg. 473-482, San Jose, 2002.

[6] Beomjoo Seo and Roger Zimmermann. Efficient disk replacement and data migration algorithms for large disk subsystems. ACM Transactions on Storage (TOS), volume 1, issue 3, pg 316-345, August 2005.

[7] A. Miranda, S. Effert, Y. Kang, E.L. Miller, A. Brinkmann, T. Cortes. Reliable and Randomized Data Distribution Strategies for Large Scale Storage Systems on 18th Annual International Conference on High Performance Computing Bangalore, India, December 18-21, 2011.

[8] D. Yao, C. Shahabi, Per-Åke Larson. Disk Labeling Techniques: Hash-Based Approaches to Disk Scaling. Technical Report, University of Southern California, 2003.

[9] S. A.Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In Proceedings of the International

Conference on Super Computing (SC'06). Tampa Bay, FL. 2006.

[10] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, distributed data placement strategies for storage area networks. In ACM Symposium on Parallel Algorithms and Architectures, pp. 119-128. 2000

[11] A. Devulapalli, D. Dalessandro, and P. Wyckoff. Data Structure Consistency Using Atomic Operations in Storage Devices. In Proceedings of the 5th International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI), pages 65 – 73, Baltimore, USA, 2008.

[12] R. J. Honicky and E. L. Miller. A fast algorithm for online placement and reorganization of replicated data. In Proceedings of the 17th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Nice, France, Apr. 2003.

[13] A. Thomasian, Y. Tang, "Performance, Reliability, and Performability Aspects of Hierarchical RAID," nas, pp.92-101, 2011 IEEE Sixth International Conference on Networking, Architecture, and Storage, 2011

[14] Y. Kwak, B. Gu, S. Cheong, J. Hwang, Y.Choi, Performance Analysis of RAID Implementations, U- and E-Service, Science and Technology, Communications in Computer and Information Science Volume 62, 2009, pp 47-52

[15] J. Gonzalez and T. Cortes, "Increasing the Capacity of RAID5 by Online Gradual Assimilation," Proc. Int'l Workshop Storage Network Architecture and Parallel I/Os, Sept. 2004.

[16] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. Ganger, "Argon: Performance Insulation for Shared Storage Servers," Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07), Feb. 2007.

[17] C. Lu, G. Alvarez, and J. Wilkes, "Aqueduct: Online Data Migration with Performance Guarantees," Proc. First USENIX Conf. File and Storage Technologies (FAST '02), pp. 219-230, 2002.

[18] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song, "PRO: A Popularity-Based Multi-Threaded Reconstruction Optimization for RAID-Structured Storage Systems," Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07), pp. 277-290, Feb. 2007.

[19] C.R Franklin and J.T. Wong, Expansion of RAID Subsystems Using Spare Space with Immediate Access to New Space, US Patent 10/033,997, 2006.

[20] Suzhen Wu, Hong Jiang, Dan Feng, Lei Tian, and Bo Mao, WorkOut: I/O Workload Outsourcing for Boosting the RAID Reconstruction Performance, In Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09), San Francisco, CA, USA, pp. 239-252. February 2009.

[21] 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 International Parallel and Distributed Processing Symposium (IPDPS'04), IEEE, 2004.

[22] K. Dasgupta, S. Ghosal, R. Jain, QoSMig: Adaptive rate-controlled migration of bulk data in storage systems. In Proceedings of the International Conference on Data Engineering (ICDE'05), 2005, 816–827.

[23] A. Verma, U. Sharma, J. Rubas, An architecture for lifecycle management in very large file systems. In Proceeding of the 22nd IEEE-13th NASA Goddard Conference on Mass Storage Systems and Technology (MSST'05), 2005.

[24] A. Brinkmann, S. Effert, F. Meyer auf der Heide, and C. Scheideler. Dynamic and Redundant Data Placement. In Proceedings of the 27th IEEE International Conference on Distributed Computing Systems (ICDCS), Toronto, Canada, June 2007.

[25] C. Schindelhauer and G. Schomaker. Weighted distributed hash tables. In Proceedings of the 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pages 218–227, Las Vegas, Nevada, USA, July 2005.

[26] 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 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2004.