Enhancements in data redistribution strategies to increase efficiency of large data volumes in Scientific Clouds using FastScale

Enkeleda KUKA, Mirela NDREU, Genti DACI, Aleksander XHUVANI

ABSTRACT – **In many scientific Clouds, storing very large amounts of application data remains a great challenge. To provide necessary storage and performance support, one strategy is to distribute data over multiple disks using RAID technologies which are widely available and very robust. Adding new storage disks to cope with large amounts of Application data, requires proper parallel data redistribution techniques to maintain the performance of entire system. In this paper we describe various techniques and algorithms that take advantage from redistribution strategies aiming on increasing the performance of a scalable parallel disk array. We will summarize several recent methods and approaches like SCADDAR, SLAS, ALV and FastScale. We will describe FastScale implementation and propose an algorithm to take in account parity block position structures to enable parallel read/writes on the extended volumes. Numerical results show that FastScale outperforms SLAS under the same workloads. We conclude with a discussion of the expected performance or proposed algorithm and future works on performance evaluation.**

KEYWORDS: RAID, SCADDAR, SLAS, FastScale, Cloud Computing.

I. INTRODUCTION

RAID systems were projected to give higher performance, more capacity and data reliability of the existing system. Especially level 5 RAID was the first one that gave us the possibility to make parallel reads and writes, thus to make an application, or a system much more efficient [1]. However, user data increases continuously and applications almost always require larger storage capacity. to supply always the needed capacity and/or bandwidth, one solution is to add extra disks to the RAID volume.

Uniform and parallel data distribution is important for RAID volumes to maintain high levels of performance, thus different approaches are developed to make the redistribution of data blocks when a disk addition is performed. We will discuss some of these approaches giving the singularity of every technique used by them.

Initially we will see SCADDAR, an efficient, online method to scale disks in a continuous media server. SCADDAR uses a series of REMAP functions which derive the location of a new block using only its original location as a basis [5]. Practically SCADDAR maintains load balancing of blocks and in general low complexity computation. During the data redistribution process, there is always a reordering window, and based on it is developed another approach. SLAS approach uses the priorities of the reordering window, 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.

The ALV approach applies the technique of the reordering window on RAID-5 storage volume and this increases the efficiency of the system [4]. From the experiments results it was concluded that ALV had a noticeable improvement over earlier approaches in two metrics: the redistribution time and the user response time. The last approach that we found is FastScale, a scaling approach that gives a very high performance on RAID-0. The strength of this algorithm derive by its addressing function which minimizes data movement [2].

Figure 1 shows the movement of data blocks on RAID-0 with FastScale. During the review of FastScale we noticed that the blocks move from old disks to the added ones, but they do not change the physical address. Our main goal is to find a solution how to improve the performance of RAID-5 using FastScale. FastScale is not implemented in RAID-5 because it has not included parity bit in its addressing function.

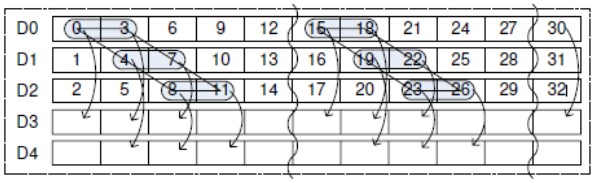

Figure 1. The movement of blocks in RAID-0 after adding two new disks using FastScale.

The contribution of this paper is the proposal of an algorithm that includes parity bits in the distribution process of FastScale. The parity block position structure enables parallel reads/writes in RAID-5 volumes the and we take account of them in our algorithm.

Practically, we do not change FastScale algorithm, but we only add a short algorithm to the normal phases of the redistribution to RAID-5. The algorithms must be implemented between data migration and parity calculation, and it does not make complex computation. This gives us the reason to predict that the performance of the system will be at high levels, as it happens on RAID-0 volumes.

II. THE DEVELOPMENT OF THE SCALING TECHNIQUES

A. SCADDAR

The first approach that we will mention is SCADDAR [5]. The disk addition to a continuous media server is still

nowadays necessary and it cannot be avoided, but the necessity was to find an approach that makes the redistribution of data without interruptions of the activity. This need motivated the development of SCADDAR which is an efficient, online method to scale disks in a continuous media server.

With SCADDAR, we are able to use pseudo-random placement without redistributing all the blocks after each scaling operation. SCADDAR does not require a directory for storing block locations, only a storage structure for recording scaling operations, which is significantly less than the number of all block locations. In addition, SCADDAR computes the new locations of blocks on-the-fly for each block access by using a series of inexpensive *mod* and *div* functions. Practically, 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. Only those blocks which need to be moved are moved and 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.

Several experiments have been performed to show that SCADDAR provides load balancing. 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, but however the improvements have not stopped with this one.

B. SLAS

Discussing on one of the most important problems in current systems, the increasing demand of applications for higher I/O performance and larger storage capacity, there are two well-known striping policies: round-robin policy and random policy. Random striping appears to be more flexible when adding new disks or deleting existing disks. But, due to its poor performance and lack of qualified randomized hash function, random striping is not so satisfactory a solution as expected. Round-robin striping is used in the most applications that demand high bandwidth and massive storage because it gives to the system uniform distribution and low-complexity computation. Round-robin striping is applied in different storage systems: 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.

These are the reasons why another approach that provides the redistribution of data after adding disks. 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. This is the basis of SLAS approach.

Figure 2 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.

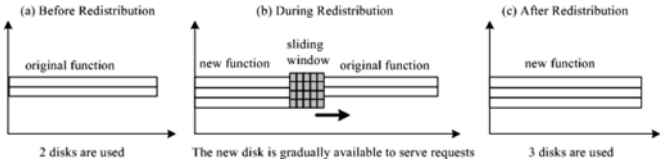

Figure 2. Mapping management based on a sliding window for the data redistribution

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.

SLAS has another characteristic: 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. However, SLAS during redistribution moves all the blocks of data into all the disks and this proved that is a weakness for SLAS. Moving all data blocks is not necessary and this reduces system performance.

C. ALV

ALV is another approach that is based on the reordering window [2]. This approach 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. The three techniques that ALV uses 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. From the experiment results it was concluded that ALV had a noticeable improvement over earlier approaches in two metrics: the redistribution time and the user response time.

There is an essential difference between RAID-0 and RAID-5 and it is predictable that this difference will influence the scaling process of each volume. In the Figure 3 are illustrated the initial states of the redistribution process in RAID-5 volume. The presence of parity blocks orients all the blocks movement. This illustration of the process demonstrates 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.

Our focus is precisely on RAID-5 volumes and this is why ALV was an interesting approach for us, but anyway we did not stop to this one. We find another approach that gives a higher performance.

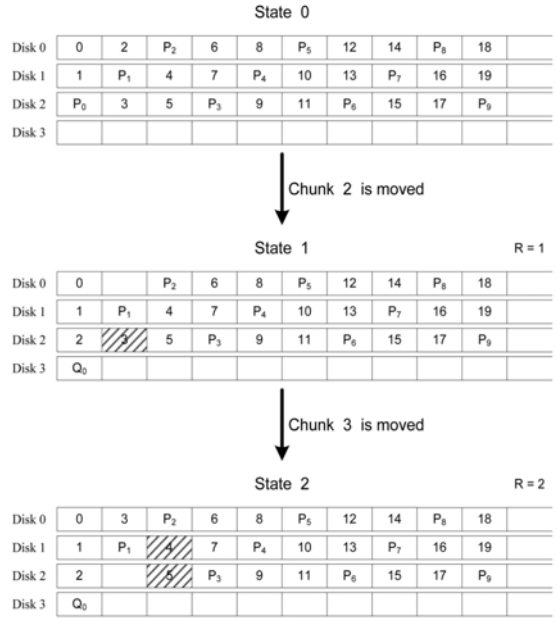

Figure 3. A series of states in data redistribution for RAID-5 scaling from 3 disks to 4. The reordering window is represented by "R".

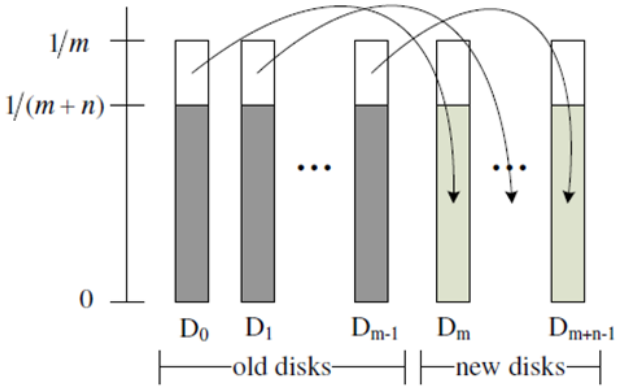

Figure 4. Data migration using FastScale. No data is migrated among old disks.

III. FASTSCALE

The last approach that we will discuss is FastScale, an approach that can tolerate multiple disk addition moving the minimum amount of data. The basic idea of the FastScale approach is shown in Figure 4.

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.

One RAID scaling process can be divided into two logical stages: data migration and data filling. In Figure 5 is shown the first stage, a fraction of existing data blocks are migrated to new disks. For the RAID-0 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 with 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.

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.

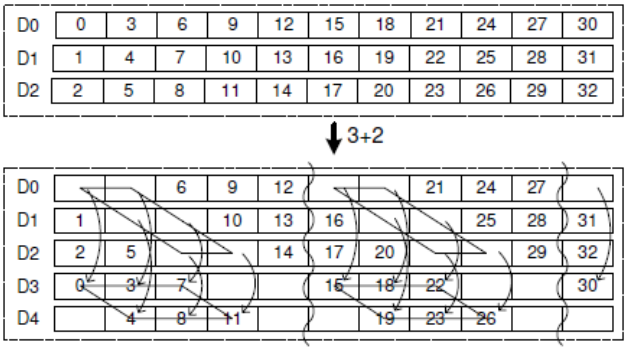

Figure 5. The data migration process in RAID-0 using FastScale.

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 6 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.

IV. OUR ALGORITHM

Our goal is to give to RAID-5 scaling a higher performance using the techniques that FastScale owns. The restriction of FastScale is that it does not include parity bits in the algorithm. 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

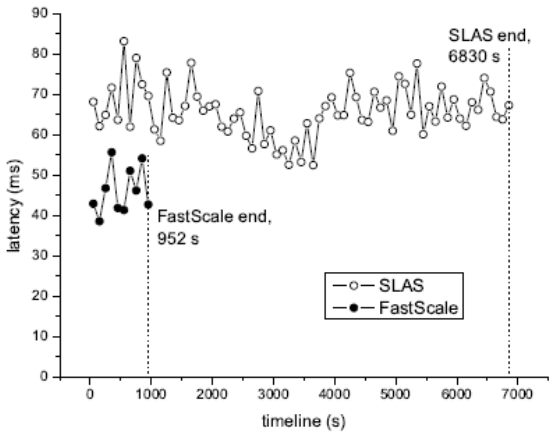

Figure 6. Performance comparison between FastScale and SLAS under the same workload.

blocks in regions. As it is shown in Figure 7, 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.

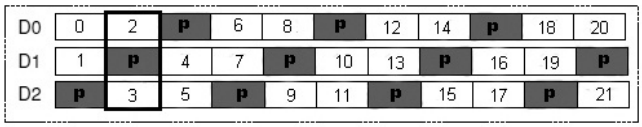

Figure 7. The structure of RAID-5 and the blocks with the same physical number.

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 data and write parity bit 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.

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

1. **if R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1][*b*] == null** // we control if the physical block is written
2. ***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

3. **ParityComputationProcedure (*d*$_0$, *b*$_0$)** // we call the procedure that calculates the new parity
4. **exit;**
5. **else if R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1] [*b*] == parity bit**
6. ***d*$_0$ □ R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1], *b*$_0$ □ *b***
7. **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

8. **ParityComputationProcedure (*d*$_0$, *b*$_0$)**
9. **exit;**
10. **else if R[(*m*+*n*) – *b*mod (*m*+*n*-1) - 1] [*b*] == info bit**
11. ***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

12. **R[*m* – *b*mod (*m* - 1) – 1][*b*] □ R[*d*$_0$][*b*$_0$]**
13. **R[*d*$_0$][*b*$_0$] □ null** // after moving info bits, the old parity is not necessary, so we can delete it
14. **ParityComputationProcedure (*d*$_0$, *b*$_0$)**
15. **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.*

In Figure 8 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.

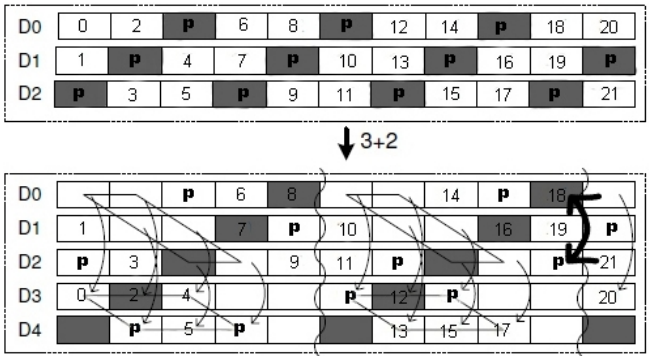

Figure 8. The redistribution process in RAID-5 using the proposed algorithm.

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.

V. CONCLUSIONS AND FUTURE WORK

Due to the fact that RAID-5 volumes give to an application more performance and effectiveness because of the supported parallel reads/writes, our attention was concentrated on RAID-5 and especially we worked on finding a solution how RAID-5 can be even more efficient. Time after time we have to scale the RAID volume with extra disks because the application data increases continuously and then we face with the problem data redistribution.
This paper exploits the general characteristics of some approaches that perform the data redistributing after scaling storage devices. FastScale is a scaling algorithm that gives a high performance in RAID-0 volumes.
The contribution of this paper is the proposal of one algorithm that can adapt FastScale to RAID-5 systems. Our proposal is a short algorithm that includes parity bits in the redistribution process of FastScale. We do not change the addressing function of FastScale algorithm, which is its main strength, but we only add a short algorithm to the normal phases of the redistribution to RAID-5. The algorithms must be implemented exactly after the migration of data and before the parity calculation and it does not make complex computation. This gives us the reason to

predict that the performance of the system will be at high levels, as it happens on RAID-0 volumes.

But, in this paper is not shown any numerical analysis or performance simulation of our algorithm, and this will be on focus for our future work.

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