# RAID Architecture with Correction of Corrupted Data in Faulty Disk Blocks

Henning Klein, Fujitsu Siemens Computers, Augsburg, Germany, henning.klein@fujitsu-siemens.com
Jörg Keller, Fernuniversität in Hagen, Hagen, Germany, joerg.keller@FernUni-Hagen.de

## Abstract

Disk capacities and processor performances have increased dramatically ever since. With rising storage space the probability of failures gets higher. Reliability of storage systems is achieved by adding extra disks for redundancy, like RAID-Systems or separate backup space in general. These systems cover the case when disks fail but do not recognize corrupted data in faulty blocks. Especially new storage systems like Solid State Drives are more vulnerable to corrupted data as cells are “aging” over time. We propose to add error detection and correction of data to a RAID-system without increasing the amount of space needed to store redundancy information compared to the common implementation RAID 6. To overcome higher computation complexity the implementation uses parallel execution paths available in modern Multicore and Multiprocessor systems.

## 1 Introduction

A hard disk drive cannot be seen as a reliable and durable medium when it comes to storing valuable data. Hard disks can fail suddenly by defects in the controller or more often in the mechanics of the disk. Therefore several techniques have been developed in the past years to overcome problems with hard disk failures. The most common technique is RAID, where one or more extra disks are used for adding redundant information in order to tolerate the loss of one or more disks. However data on hard disks can get corrupted without a disk failure. This issue has dramatically increased since the introduction of solid state drives, where it is only a matter of time and usage frequency when cells are unable to store data correctly. Data on storage drives is checksum protected. Therefore the host controller usually gets notified if corrupted errors are detected. These “noisy” errors can then be corrected by recovering the data from redundant data stored on different disks of the storage array. However, a lot of components handle the data until it is finally stored in RAM and each one can fail without recognition. These failures are also called “silent” errors and have already been investigated in an experiment. For further information see [3]. Several techniques, from hardware to software levels, have been investigated to detect and recover corrupted data. The disadvantage of these schemes is that additional storage is needed for detecting errors and in most cases a new storage independent layer is implemented. We propose a technique for extending a RAID 6 system to detect and correct corrupted data in faulty disk blocks, while using the same amount of storage space as the common RAID 6 system. Our approach provides the same level of data reconstruction in case of hard disk losses and allows the correction of more combinations of noisy data block errors. In addition our system also has the ability to detect up to three defect data blocks (silent errors) in a certain range and correct it, so the probability of undetected corrupted data can be dramatically decreased. RAID 6 uses two parity values and can therefore only correct one faulty block and only if all disks are working. The system used in our proposal offers the capability of correcting single blocks in a certain range even after one disk stopped working. The proposed technique involves more complex computations, based on Reed Solomon codes, than those without error detection. Therefore a parallel implementation is introduced which is optimized for latest multicore and multiprocessor systems. Validation of this implementation demonstrates that the error correction ability works and that the performance on standard desktop computers is sufficient for storage systems using the latest disk drives.

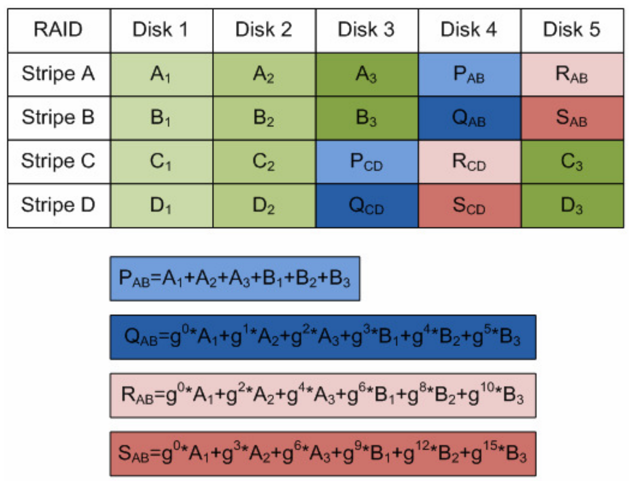
The remainder of this paper is structured as follows. Section 2 described related work. In Section 3 the proposed technique and algorithm are introduced. Section 4 validates the performance and error correction capability of an implementation. Section 5 gives a conclusion and an outlook on future work.

## 2 Related Work

RAID (Redundant Array of Inexpensive Disks) is a common technique used to increase performance, reliability or both. It has been introduced in [6]. Among the variety of RAID systems a disk array tolerating the failure of two disk losses, called RAID 6 as described in [7] has been investigated, which is able to tolerate two disk losses with two additional disks using Reed Solomon codes. Other techniques followed in order to overcome the computation complexity when Reed Solomon codes are used: EVENODD [2] and DATUM [1] are only two of numerous examples. If too many errors are encountered, a Reed Solomon decoder may miscorrect to another code word with a certain probability [9], which also constitutes a kind of error. To solve the problem of undetected errors, systems have been proposed storing separate values for checksums like [8] or ZFS of [10] amongst others. The proposed technique combines the advantages of both approaches, i.e. it is able to detect and correct corrupted data and tolerates the loss of two disks without the need of extra storage compared to RAID 6. RAID systems [4] and Reed Solomon codes [5] have also been accelerated by using configurable hardware. While this seems a viable option in systems-on-chip and in high performance computers like the Cray XD1, our approach targets standard personal computers where we cannot expect reconfigurable hardware, for which reason we have focussed on a software solution.

## 3 Concept

The proposed technique is based on Reed-Solomon correction codes like in the RAID 6 system. Like the well known RAID it uses two hard disk drives for storing redundant data (parity) and is able to recover data in the case of the loss of up to two hard disks. Just like in RAID 6 systems the content of each disk drive is divided into equally sized blocks. One row of blocks across all disks in an array having the same offset is called a stripe. Two blocks of one stripe contain parity information instead of data. The parity information is not stored on two specific disks. It is spread across all disks, changing the position every two stripes, see blue and red blocks in Figure 1. Instead of calculating two parity values like in RAID-6 the proposed system uses four values P, R, Q and S. The parity values are computed using the data across two stripes. Figure 1 shows an example of the algorithm for an array of five disks.



| RAID | Disk 1 | Disk 2 | Disk 3 | Disk 4 | Disk 5 |
|---|---|---|---|---|---|
| Stripe A | $A_1$ | $A_2$ | $A_3$ | $P_{AB}$ | $R_{AB}$ |
| Stripe B | $B_1$ | $B_2$ | $B_3$ | $Q_{AB}$ | $S_{AB}$ |
| Stripe C | $C_1$ | $C_2$ | $P_{CD}$ | $R_{CD}$ | $C_3$ |
| Stripe D | $D_1$ | $D_2$ | $Q_{CD}$ | $S_{CD}$ | $D_3$ |

$P_{AB}=A_1+A_2+A_3+B_1+B_2+B_3$

$Q_{AB}=g^0*A_1+g^1*A_2+g^2*A_3+g^3*B_1+g^4*B_2+g^5*B_3$

$R_{AB}=g^0*A_1+g^2*A_2+g^4*A_3+g^6*B_1+g^8*B_2+g^{10}*B_3$

$S_{AB}=g^0*A_1+g^3*A_2+g^6*A_3+g^9*B_1+g^{12}*B_2+g^{15}*B_3$

**Figure 1** - Position of parity and data blocks

The algorithm for parity generation is based on the Galois Field GF($2^8$) with generators $g^n$ that are multiplied with data values of the disks. This enables the reconstruction of four data blocks within a set of two stripes, if the position of these disks is known. If disks are defect the missing blocks are known and can therefore be recovered.

Corrupted data in data blocks can be determined by recalculating and comparing parity values. If the parity value on disk and the calculated one do not match, up to three blocks can be reconstructed. The probability of a successful recover depends on the number of blocks that have been corrupted and the number of disks. See Section 4 for more details. There are three different scenarios when parity calculations have to be done: Generating parity before writing, checking parity after reading and recovering data after losing one or two disks. The system can be used in a similar way for disk arrays tolerating a different number of disk losses than two.

## 4 Validation of performance and error correction capability

The proposed algorithm was implemented to be able to perform experiments about error correction capability and performance. RAID arrays are used to increase storage performance and can reach transfer rates at a multiple speed of a single disk. Therefore the algorithm for parity computation is parallelized and optimized for modern multicore systems with high cache capacity. Finite field multiplications and divisions are complex operations which need large numbers of instructions on x86 processor architectures. Therefore lookup tables are initially generated to speed up those operations on constants. In this case 3.3 Kilobytes of tables are required to perform a full recovery of four blocks in two stripes without the need of multiplications or divisions. The only operations left are XOR-computations. After parallel implementation the performance of two main operating modes has been measured with a quadcore processor system[1]. One typical mode is the computation of the parity. Figure 1 shows the performance when using one to four processors. The parity consists of four values that have to be generated when storing data. After data has been read these values can be computed again and compared to make sure the data has not changed. This process can be accelerated if less than four parity values are computed. However, the probability of undetected defect blocks will increase.

If one or two disks fail data has to be recovered. If only one disk fails data integrity checks can still be done and one corrupted block in two stripes can still be corrected. Figures 2 and 3 show the performance scaling from one to four processors when data of one or two disks is recovered. In worst case, after two disks are lost, the system provides up to 150 MB/s utilizing one and almost 600 MB/s using four processors. If one disk is lost the speed goes up to 300 MB/s on one and up to 800 MB/s on four processor systems. If integrity checking is used some overhead has to be added in order to calculate integrity checks with the remaining two parity blocks.

[1] Fujitsu Siemens Computers Amilo Pi3630, Intel Core 2 Quad Prozessor Q9400

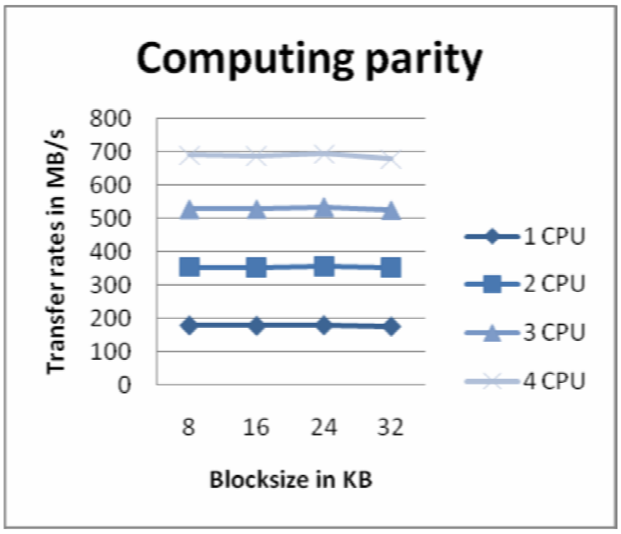


**Figure 2** - Parity computing performance

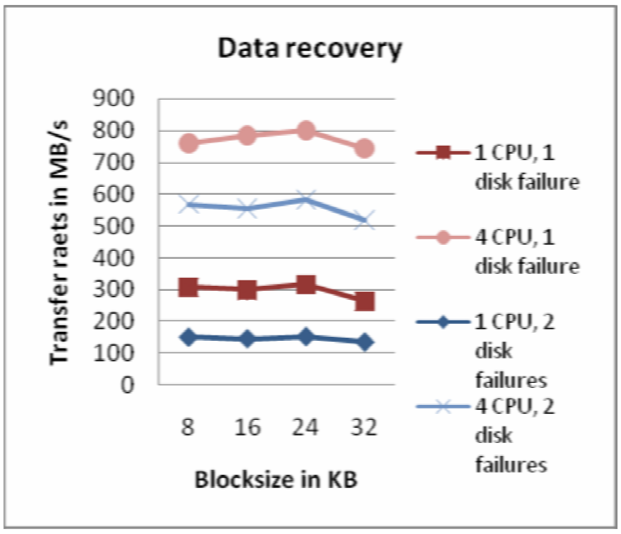


**Figure 3** - Data recovery performance

RAID-6 is able to detect two and correct one silent error within one stripe. If one disk fails single errors can be detected but not corrected. Our solution is able to detect up to four and correct up to three silent errors within two stripes. If one disk fails two errors can be detected and one error can be corrected.

When correcting silent errors the success of a correction depends mostly on the number of concurrent errors within two stripes and the number of disks in the array. If both parameters increase the number of combinations of possible corrupted data bytes grows as well. To show the capabilities of our system, pseudo random data has been generated and parity values computed. In 100.000 test cycles one, two and three errors have been simulated on pseudo random positions. After that an attempt to reconstruct the data has been made. If one or two errors were injected, the data has always been constructed correctly. Figure 3 shows the capability of corrections when three simultaneous errors occur. Triple errors are already more likely to be recovered wrongly than correctly when eight data drives are used. This is because the location of the error is unknown. However, the probability of triple errors within two stripes is low and is just in case only one byte at the same position within three blocks in two stripes of at least two disks got corrupted. If more than one byte in a block has been corrupted, for example a whole sector, the probability of a successful correction can be increased as the same combination of predicted erroneous positions can be checked when other bytes are recovered.

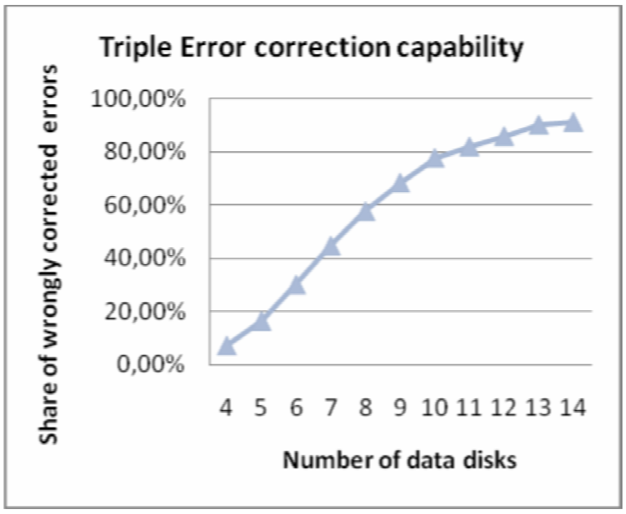


**Figure 4** - Frequency of wrong error correction

Compared to RAID-6 our proposal allows the correction of more noisy error constellations, too. Within two stripes each system allows the correction of a total of up to four errors. In our solution four errors can be corrected even if all of them are within one stripe. RAID-6 only allows the correction of two errors per stripe. Figure 5 shows the number of correctable error combinations within two stripes for both systems if all disks are running and after losing one disk.

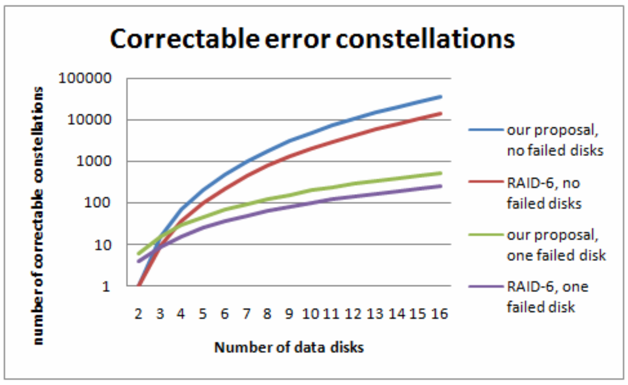


**Figure 5** - Correctable error constellations

## 5 Conclusions and Future Work

We introduced a technique which can be used to overcome data corruptions on disks, especially on newer disk systems like Solid State drives which are more likely to fail because of cells wearing out or not being accessed for a longer amount of time. Utilizing this RAID technology a fast and reliable array of SSDs consuming little power can be constructed. The proposed scheme can easily be changed tolerating any other number of disk failures while being able to detect and correct data corruptions. A system tolerating one disk failure has already been investigated. By integrating data encryption or applying a diffusion filter the ability to detect triple errors can be improved to almost 100%.

## 6 Literature

[1] Alvarez, G. A.; Burkhard, W.A. and Cristian, F.: Tolerating Multiple Failures in RAID Architectures with Optimal Storage and Uniform Declustering. In Proc. 24th International Symposium on Computer Architecture, pp. 62-72, 1997.

[2] Blaum, M., et al.: EVENODD: An optimal scheme for tolerating double disk failure in RAID architectures. IEEE Transactions on Computers Vol. 44 No. 2, pp. 192-202, 1995

[3] Bonwick, Jeff; Moore, Bill: ZFS – The last word in File Systems. [online] http://www.opensolaris.org/os /community /zfsopensolaris.org/os/community/ zfs/docs/zfs_last.pdf

[4] Gilroy, M.; Irvine, J.: RAID 6 Hardware Acceleration. In Proc. International Conference on Field Programmable Logic and Objects, pp. 1-6, 2006

[5] Hampel, Volker; Sobe, Peter; Maehle, Erik: Experiences with a FPGA-based Reed/Solomon-encoding coprocessor. Microprocessors & Microsystems Vol. 32 No. 5-6, pp. 313-320, 2008

[6] Patterson, David A.; Gibson, Garth and Katz, Randy H.: A Case for Redundant Arrays of Inexpensive Disks (RAID). In Proc. International Conference on Management of Data (SIGMOD), pp. 109-116, 1988

[7] Plank, J.: A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Software Practice and Experience Vol. 29 No. 9, pp. 995-1012, 1997

[8] Sivathanu, G.; Wright, C. P. and Zadok, E.: Ensuring Data Integrity in Storage: Techniques and Applications. Proc. 2005 ACM workshop on Storage security and survivability (StorageSS'05), pp. 26-36, 2005

[9] Sofair, Isaac: Probability of Miscorrection for Reed-Solomon Codes. Proc. International Conference on Information Technology, Coding and Computing (ITCC'00), pp. 398-401, 2000

[10] Sun Microsystems: Sun On-Disk Specification. [Online] http://opensolaris.org/os/community /zfs/docs/ondiskformat0822.pdf