Simulating fault injection on disk arrays

Henning Klein, Fujitsu Technology Solutions, Augsburg, Germany
Jörg Keller, Fernuniversität in Hagen, Hagen, Germany

Abstract

We present an application for the simulation of errors in storage systems. The software is completely parameterizable in order to simulate different types of disk errors and disk array configurations. It can be used to verify and optimize error correction schemes for storage. Realistic simulation of disk errors is a complex task as many test rounds need to be performed in order to characterize the performance of an algorithm based on highly sporadic errors under a large variety of parameters. The software allows different levels of abstraction to perform quick tests for rough estimations as well as detailed configurations for more realistic but complex simulation runs. We believe that this simulation software is the first one that is able to cover a complete range of disk error types in many commonly used disk array configurations.

1 Introduction

In the past few years a lot of investigation regarding storage reliability has been done. In [10] the influence of several parameters like utilization, logged SMART errors or temperature on disk failure have been analyzed. In some cases only part of the disk fails which is called a visible error if the hard disk is able to detect it. In [1] 1.53 million disks have been monitored to get an insight about latent errors which remain undetected by the disk. In these papers, as well as in [5] and [11], figures are presented describing probabilities of multiple error types depending on a variety of parameters.

In order to overcome these errors several techniques have been proposed. RAID as described in [1] is being used to tolerate total failure of one or multiple disks, depending on the configuration. Visible disk errors can be corrected that way, too. We proposed a new scheme to detect and correct multiple latent errors in a disk array in [6]. A different attempt to tolerate latent errors on single disks using intra disk redundancy has been presented in [2].

In order to prevent multiple latent errors from piling up, which would reduce the chance of a successful correction, data has to be checked frequently. By "scrubbing" the disk, the data is being read out in order to search for errors. However, additional IOs increase the probability of errors and entail higher power consumption. Therefore several attempts have been made to optimize scrubbing techniques, cf. e.g. [7], [8], [10].

Validation and comparison of scrubbing algorithms and redundancy schemes with respect to the reduction of latent errors, performance and/or power consumption is not feasible analytically and costs effort if done by simulation (see related work). We therefore present a simulator for disk storage systems including means to simulate faults and scrubbing measures. The simulator allows multiple levels of detail, and thus enables both quick but rough estimations and complex realistic scenarios. Furthermore, the simulator is parameterizable to allow for different error models, disk configurations and disk usage patterns. It allows the definition of block level reliability as a function of parameters like utilization and age. This way the reliability of data cells in Solid State Disks can be simulated. However, due to a lack of information regarding NAND cell reliability of SSDs and the new technology that is changing rapidly the simulation of SSDs will be discussed in our future work.

The remainder of this paper is organized as follows. In Section 2 we give a brief overview about Related Work. Section 3 describes the architecture of the simulation software. In Section 4 we compare simulation results with measured values, present performance results, and discuss simulation results of a self-correcting and self-checking disk architecture. In Section 5 we give a conclusion and an outlook on future work.

2 Related work

In previous work, we have proposed a variant of the RAID-6 redundancy scheme optimized for silent errors, and we have proposed a scrubbing algorithm removing those errors in a disk system. We present this work in Section 3.2.

Validating redundancy schemes in a real world scenario is ineffective as errors are highly sporadic and depend on various factors like disk vendor, type and batch as well as the age, utilization and the operating environment. Therefore the probability is either calculated or simulated using proprietary tools for specific scenarios like in [8] because general purpose disk simulators like DiskSim [3] would be too slow to run several thousand test runs that are necessary to get significant average values. Even Ram Disks like [4] are too slow as they have to store I/O values.

The proposed simulator is able to work in different levels of detail and therefore is able to generate quick results to compare and optimize scrubbing algorithms in specific scenarios as well as test them in complex disk arrays with different sets of parameters.

3 Simulator architecture

3.1 Supported configurations

The proposed tool simulates the behavior of a disk array (RAID) over an extended period of time and uses fault injection to allow validation of the robustness against multiple kinds of errors. The simulator supports single disks as well as the well known RAID levels RAID-5 and RAID-6. We have also implemented our proposed RAID architecture with latent error detection and correction capabilities [6]. Extension to other redundancy schemes, i.e. MDS codes, should be possible with restricted effort.

Once the disk architecture is set up, optional events can be added and customized to detail the simulation process. Depending on the complexity and frequency of the occurrence of an event the runtime of a simulation process will be slowed down.

The application allows adding disk accesses which can be modified to match realistic patterns, i.e. generating hot spots like cluster bitmaps in file systems or data accesses that decrease in frequency over time like in data archives.

We have implemented three different scrubbing algorithms: sequential scan, staggered scrubbing as proposed in [8] and our proposal [7] which prioritizes areas that are rarely accessed. The simulator allows defining multiple scrubbing schemes that are enabled or disabled by certain events such as disk age or detected faults.

The tool supports injection of various error types. Error events can be configured to overlap. That way simultaneous disk faults can be generated. However, double faults such as a latent error on a failed disk are filtered out automatically. The tool supports four kinds of errors with configurable probabilities. Dependency factors like disk age or utilization can be integrated. If a total disk failure is simulated in a redundant RAID array, the time to replace the disk and the speed at which the RAID rebuilds parity or data information can be specified. Before the rebuild process completes, data on the new disk is handled like a visible error section that is being reconstructed sequentially. Reconstructed data can therefore be used to recover data on other disks even though the rebuild process hasn't completed yet. Visible and latent errors can be specified in probability, size and position. As shown in [1] latent errors often spread across the disk. Therefore we have implemented the option to specify the amount, position and size of subsequent errors. If a visible error occurs, an immediate attempt is being made to recover lost data. Latent errors are only reconstructed after being detected. We assume that if some kind of error checking mechanism is implemented, an error is detected when data at its position is being accessed. As the location of latent errors is unknown, the data reconstruction process is more complicated and leads to unrecoverable data more often. The fourth kind of error that is supported is a data cell error that could occur on a solid state drive. The data loss of rarely accessed cells can be simulated as well as wear-out effects on highly utilized cells. This kind of error can be used to simulate error injection on solid state disks. However, currently only rough estimations are available about the reliability of NAND cells, especially about their ability to hold data over a long period of time. Figure 1 gives an overview about the modules of the simulator.

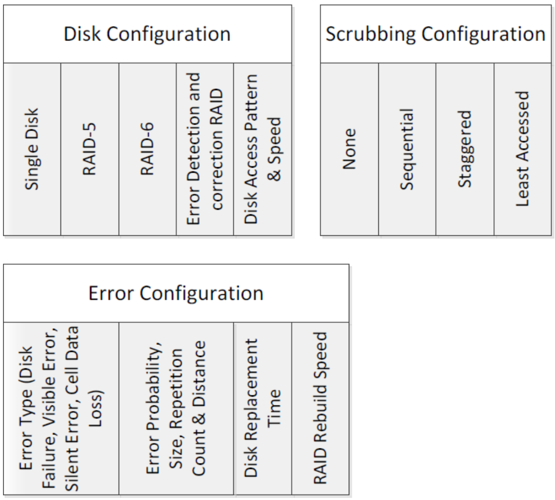

Disk Configuration
Single Disk
RAID-5
RAID-6
Error Detection and correction RAID
Disk Access Pattern & Speed
Scrubbing Configuration
None
Sequential
Staggered
Least Accessed
Error Configuration
Error Type (Disk Failure, Visible Error, Silent Error, Cell Data Loss)
Error Probability, Size, Repetition Count & Distance
Disk Replacement Time
RAID Rebuild Speed

Figure 1 Configuration overview

3.2 Simulated RAID and disk scrubbing methods

The simulation software implements a couple of common RAID configurations (Level 0, 1, 5, 6) as well as our proposal, a variation of RAID-6. We also analyze a RAID-5 variation which matches the original proposal in degraded mode. This means that one of the disks failed and has not been replaced. Figure 2 shows the original proposal. Each disk is divided into equally sized blocks. All blocks at the same offset of each disk in the array form a stripe. Each stripe consists of data blocks and two blocks holding parity values. Our proposal differs from RAID-6 by dividing a single stripe into two parts and computing four different parity values. This method allows detection of four and correction of up to three silent errors. Figure 2 depicts the architecture. In Figure 3 the computations based on Reed Solomon Codes are given.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
Stripe 0	0	1	2	P0	R0
Stripe 1	3	4	5	Q0	S0
Stripe 2	6	7	P1	R1	8
Stripe 3	9	10	Q1	S1	11

Figure 2 RAID-6, optimized for latent error correction

$$P = D_0 + D_1 + \cdots + D_n$$
$$Q = g^0 * D_0 + g^1 * D_1 + \cdots + g^n * D_n$$
$$R = g^0 * D_0 + g^2 * D_1 + \cdots + g^{2n} * D_n$$
$$S = g^0 * D_0 + g^3 * D_1 + \cdots + g^{3n} * D_n$$

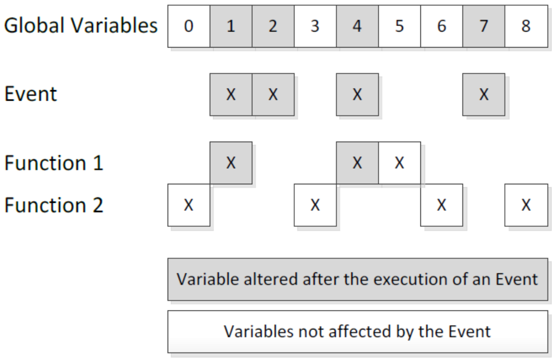
Figure 3 Parity calculation

We have proposed a new scrubbing algorithm in [7] which takes user accesses into account. Areas that are accessed frequently are being left out in the scrubbing process. Large areas that have not been checked within a certain time frame get higher priority.

The simulator implements constraints to check for uncorrectable error combinations for each RAID system. It allows the definition of the time until failed disks are replaced as well as the speed the RAID rebuilds.

3.3 Performance optimizations

The proposed simulator has been implemented in C++ for performance reasons, based on an event driven model to simulate disk access, multiple types of errors or disk scans. The execution order of each event is defined by a timing function. Each event is defined by a couple of functions, depending on the type of event. In order to avoid unnecessary computations each function is only computed if a variable the function depends on has changed. Each variable has a global scope and is assigned to a designated bit position in a 64-bit array. If a variable has changed the accordant bit is set which allows the comparison with a second static bit array that defines dependencies of a specific function. This process is depicted in Figure 4.

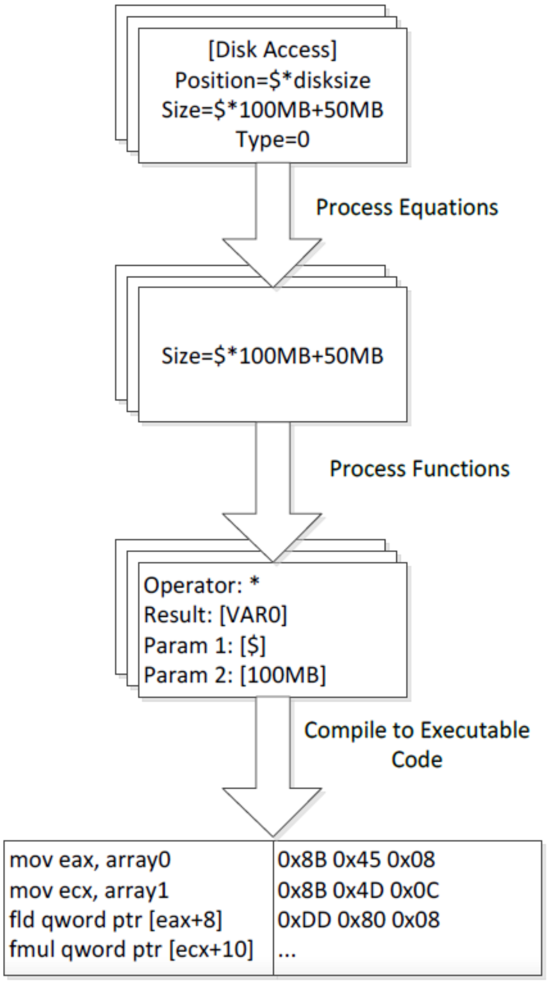


Global Variables	0	1	2	3	4	5	6	7	8
Event		X	X		X			X	
Function 1		X			X	X			
Function 2	X			X			X		X

Variable altered after the execution of an Event
Variables not affected by the Event

Figure 4 Checking for affected functions

In order to calculate these user defined parameters, millions of operations have to be processed during the runtime of a RAID simulation to calculate the parameters of the events. The user supplied string has to be parsed into variables, constants and operators in order to calculate the result. In order to get the maximum performance, we compile the code into directly executable functions stored in memory with execution rights. We dissect each equation into simple operations consisting of a result and a maximum of two arguments first. Temporary results, variables and constants are assigned to fixed positions within data arrays. Each operation can therefore be defined as a function, combined with the information of the array selection and position that holds an argument or temporary result. We defined two arrays, one that holds the global variables used in all events as well as a local array holding constants and temporary results. After a function has been parsed and compiled, all array positions are fixed. Therefore we initially have to declare all global variables used to ensure fixed array positions. Operations that can be computed in advance, e.g. operations on two constants, are solved in advance during the parsing steps. Figure 5 illustrates the compilation process.



```
mov eax, array0            0x8B 0x45 0x08
mov ecx, array1            0x8B 0x4D 0x0C
fld qword ptr [eax+8]      0xDD 0x80 0x08
fmul qword ptr [ecx+10]    ...
```

Figure 5 Compiling user defined functions

The simulation is based on tests with random numbers. Therefore multiple runs have to be performed to compute average numbers based on the results to draw conclusions on the effectiveness of the analyzed scrubbing algorithms. Additionally the simulation of RAID storage including read/write accesses has a high computation complexity. Simulating two years of a storage array with "staggered scrubbing" roughly takes 1.25 minutes to complete on a modern processor. Therefore we have parallelized the execution of simulation runs. In order to gain the best performance we run the same test definition file on multiple cores at a time. We are putting a lot of effort into the preparation of the test run, i.e. by scanning for affecting variables as well as parsing and compiling functions. By running the same test definition multiple times, we can share the prepared test environment across all parallel test executions. As tests slightly vary in the execution time, i.e. by different scrubbing rates due to randomly generated errors, the thread that completes the test run first can already prepare the environment for the next test run. Figure 6 illustrates this process.

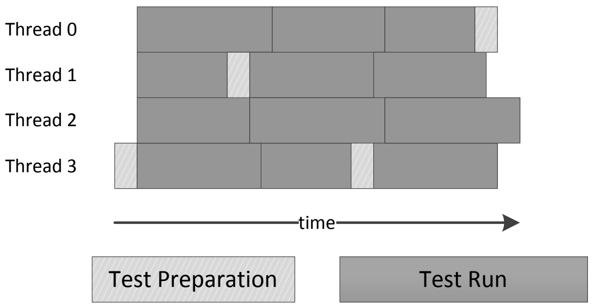


Figure 6 Multiple errors affecting one stripe

4 Simulation accuracy and performance

4.1 Simulation accuracy

The basis for our parameter settings are investigations about disk failures [10] and latent errors [1]. We use data of figure 2 of [10] to simulate total disk failures and results shown in figures 1, 4, 5 and 7 of [1] to define the probability, number, size and position of latent errors. We have imported the values of the presented charts into a test definition file by function interpolation using Mathematica. The error behavior of each disk can then be imitated by using a specific set of functions. The accuracy depends on the complexity of the polynomial.

In [10] an analysis about the probability of a total disk loss is being presented. The authors show the "mortality" rate of hard disks in dependence of age. They discovered an increased rate of failures during the first three months. In a long term, however, the drives seemed to run quite stable with failure rates around 6-9%. Figure 7 shows the comparison between simulated and original test results.

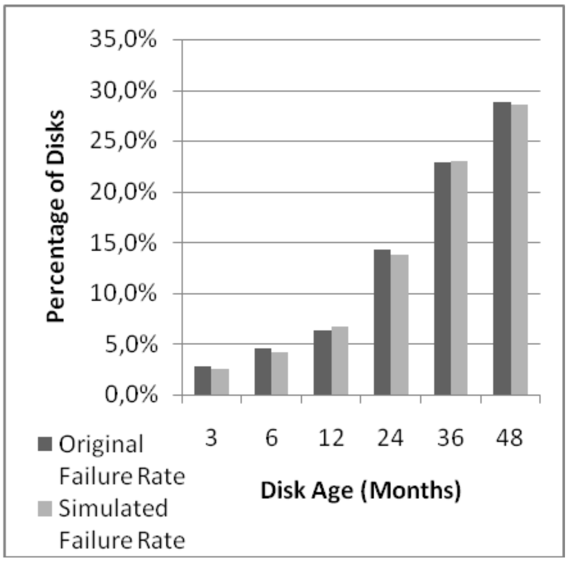


Figure 7 Disk failure rates

We have defined the behavior of each disk regarding latent sector errors as functions of occurrence probability, quantity, position and inter-arrival rate according to [1]. We ran several thousand randomized tests to get average values that can be compared to the test results of the paper. For clarity reasons we have only included two types of disks in the figures: k1, an enterprise level disk and C1, a nearline or consumer grade disk. All results of the simulation match the original values of the paper quite close. Keeping in mind that the behavior of disks of the same class differ quite strong, a failure rate of a few percent is enough for simulation purposes.

Figure 8 depicts the percentage of disks developing at least one latent error within two years. Figure 9 shows that disks which already developed a latent error are likely to produce multiple errors. These are likely to spread within physically close positions in a short timeframe, as indicated in Figures 10 and 11.

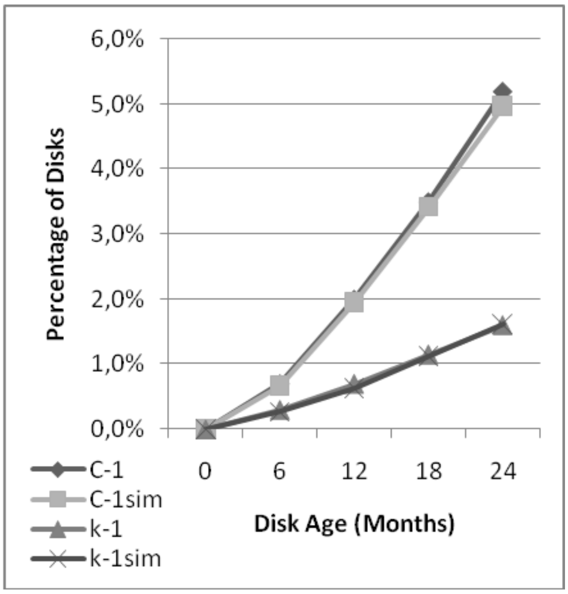


Figure 8 Latent error rates

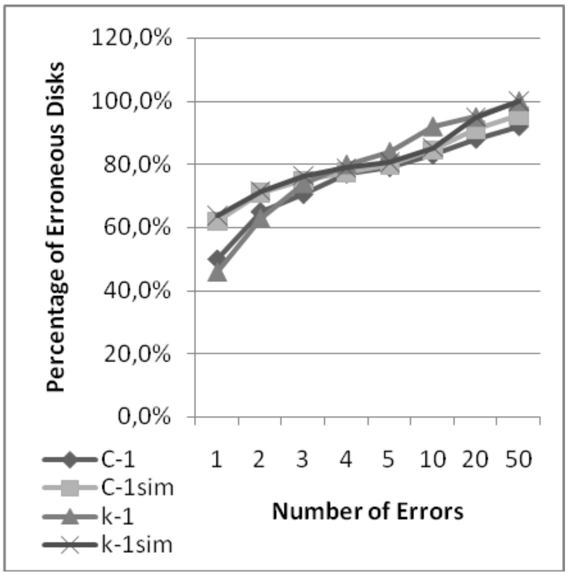


Figure 9 Consecutive latent errors

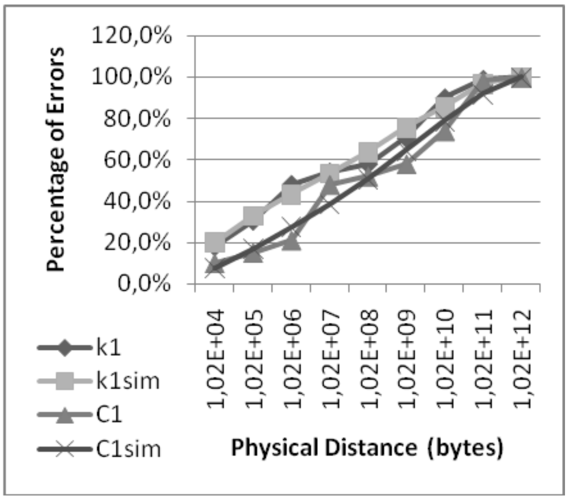


Figure 10 Physical distribution of latent errors

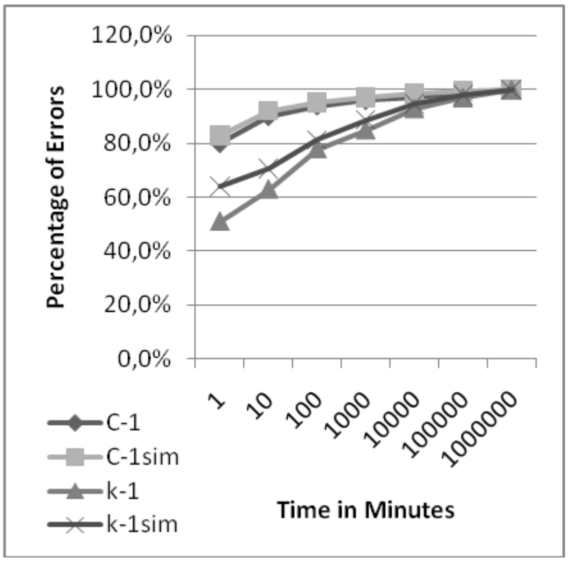
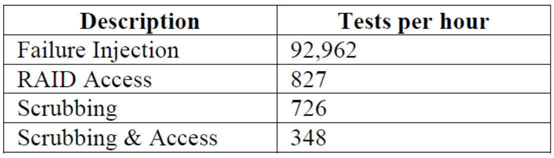


Figure 11 Interarrival rate of latent errors

4.2 Simulator Performance

In this section we give a brief overview on how the simulation software performs in different levels of detail. The basis of our test definition is a 250 GB hard disk with two types of parallel error injection: total disk failure and latent error injection. We started with the error injection on a single disk only which would suffice for analyzing the accuracy of the simulation compared to the experimental results as shown in the previous section. In the second test run we added random read and write user accesses in 10 MB blocks, which would result in the detection of corrupted data when trying to access data where a latent error has been injected previously. We changed the setup to a self-correcting RAID architecture with five disks as presented in [6]. We added a disk replacement delay of 24h and a rebuild rate of 50 GB/h in case a disk failed. Furthermore we could have altered the error probability in dependence of the utilization. The simulation results demonstrate the error correction capabilities of the proposed RAID system. In the last test we added a scrubbing algorithm which should show better results as latent errors would be detected earlier. This reduces the risk of multiple errors to overlap. Table 1 shows the number of test rounds the simulator ran per hour on a single PC using an Intel Xeon System with 3,16 GHz and four cores. Higher levels of detail demand for more computation power.



Description	Tests per hour
Failure Injection	92,962
RAID Access	827
Scrubbing	726
Scrubbing & Access	348

Table 1 Performance test results

4.3 Simulation of a self-correcting RAID

We have configured a test definition to simulate a combination of our self-correcting RAID as shown in [6] and a scrubbing algorithm presented in [7]. The self correcting RAID is a variation of a common RAID-6 system: it tolerates the loss of two disks and is additionally able to detect and correct up to three corrupted blocks per data stripe. A stripe is a set of equally sized data blocks, each of which is stored at the same physical offset on a different disk. However, the error correction capabilities are reduced after a disk failed until it is replaced and the data recovered. We reduce the risk of latent errors that could be present at the same time by frequent integrity checks (scrubbing). We chose the behavior of disk E-2 with the highest rate of latent errors from [1] and integrated the disk failure rates of [10]. The scrubbing algorithm we chose uses the same scrubbing rates as discovered in [8]. It prioritizes rarely accessed regions on the hard disk and spreads scrubbing positions evenly across the disk to increase the chance of an early detection of latent errors spreading across multiple sectors.

So far we ran over 20.000 test cycles without a case of any combination of uncorrectable errors. In 63% of all tests that were simulating a disk array running for two years, errors like disk failures or latent errors occurred that were corrected successfully. In a second test setup, we compared RAID-6 arrays with modified RAID-5 systems. We changed the latter version by doubling the stripe length and number of parity blocks. This matches the RAID-6 modification of Figure 2 if one disk failed permanently. Both systems have been tested with equal amounts of usable disk space: The original RAID-6 includes five disks whereas the modified RAID-5 consists of four disks. A reduced number of disks decreases the probability of a disk failure. We ran 2500 tests per configuration, each of which with and without our proposed scrubbing algorithm. Data loss occurs in both array types, if a latent error remains undetected until a hard disk fails. Table 2 shows the results of the two year simulations.

RAID Type	Scrubbing	Percentage of test runs with data loss
RAID-5en	NO	3,1 %
RAID-5en	YES	0 %
RAID-6	NO	8,9 %
RAID-6	YES	0 %

Table 2 RAID reliability test results

5 Conclusions

We have presented the design of an application to simulate RAID storage systems. We have discussed how the simulation software can be optimized and executed effectively on a multi-core PC. We use this simulator to evaluate scrubbing algorithms for long-term disk storage. Our results indicate that the performance of the simulator scales well with the number of processor cores. We have demonstrated that the fault injection matches the measured results of real tests accurately. In our future work we will simulate SMART events to alter scrubbing behavior or trigger early disk replacements. The current version includes common RAID Levels 0, 1, 5 and 6 and our variations. Future versions will include other MDS codes for comparison.

6 Literature

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy and J. Schindler, “An analysis of latent sector errors in disk drives”, In Proceedings of the 2007 SIGMETRICS

Conference on Measurement and Modeling of Computer Systems, 2007

[2] A. Dholakia, E. Eleftheriou, X.-Y. Hu, I. Iliadis, J. Menon, and K. Rao, "Analysis of a new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors.", ACM SIGMETRICS Performance Evaluation Review, Saint Malo, June 2006

[3] G. Ganger, “The DiskSim Simulation Environment”, http://www.pdl.cmu.edu/DiskSim/

[4] Gavaskar, V. N., “Method and apparatus for virtual disk simulation”, US Patent No. 5987565

[5] J. Gray and C. van Ingen, "Empirical Measurements of Disk Failure Rates and Error Rates", Microsoft Research Technical Report MSR-TR-2005-166, 2005

[6] H. Klein and J. Keller, "Optimizing a Highly Fault Tolerant Software RAID for Many Core Systems", International Conference on High Performance Computing & Simulation (HPCS 2009), Leipzig, June 2009

[7] H. Klein and J. Keller, " Optimizing RAID for Long Term Data Archives", Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 2010), Atlanta, April 2010

[8] A. Oprea and A. Juels, "A clean slate look at disk scrubbing", http://www.rsa.com/rsalabs/staff/bios/aoprea/publications/scrubbing.pdf

[9] D. A. Patterson, G. Gibson, and R. H. Katz, “A case for redundant arrays of inexpensive disks (RAID),” in SIGMOD ’88: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, 1988, pp. 109–116.

[10] E. Pinheiro, W. Weber and L. Barroso, “Failure trends in a large disk drive population”, FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies, 2007

[11] B. Schroeder and G. Gibson, “Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?”, 5th USENIX Conference on File and Storage Technologies, 2007

[12] G. Wang, A. Raza Butt and C. Gniady, “On the Impact of Disk Scrubbing on Energy Savings”, HotPower, 2008