THE ANNALS OF "DUNĂREA DE JOS" UNIVERSITY OF GALATI
FASCICLE III, 2012, VOL. 35, NO. 2, ISSN 1221-454X
ELECTROTECHNICS, ELECTRONICS, AUTOMATIC CONTROL, INFORMATICS

SPREAD DATA STORAGE - A MEANS OF STORING IN INTERNET

Marcel Danilescu[1]

Viorel Mînzu[2]

[1]University "Dunărea de Jos" Galați

[2]University "Dunărea de Jos" Galați

Abstract. Recent years have led to the development of new data storage solutions, moving their local storage, to the storage in cloud (Internet). In this case, data loss raises greater problems, since there it is not able to seek recovery in time due to the impossibility of direct access to hardware for storage. The purpose of this paper is to present a method of data storing that allows them availability and recovery.

Keywords: data recovery, master/slave node, parity, algorithm, storage.

1. CONCEPT AND TERMS

One of the main requirements of a computing system is the fault tolerance. This involves data availability, accuracy and possibility of their recovery in case of software or hardware failure (failure of the storage media or interruption of connections with storage media).

In general, Internet data storage is beneficial, because of creating an environment that allows remote backup data recovery in the presence of failure or local disaster.

In making storage of local data in a node in the Internet, there are involved the following elements:

- the connection between client and node.
- hardware equipment in the node.
- software (operating system and applications) of node

The occurrence of damage to any of these elements that contribute to the backup process, leads to blocking of storage service.

Over the time, there were research about local data storage, developing data recovery technologies in case of accurance to any breakdown in storage hardware, achieving different architectures and different recovery algorithms, such as :

- RAID architectures (David A. Patterson, 1998) (Michael Stonebraker Gerhard A, 1990)
- simple parity algorithm (Moon, 2005) (Morelos-Zaragoza, 2006)
- Hamming code (Moon, 2005) (Morelos-Zaragoza, 2006)
- Reed-Solomon code (Moon, 2005) (Morelos-Zaragoza, 2006)
- Double Diagonal Parity (Chris Lueth, 2010)
- EvenOdd (Mario Blaum, 1995)
- Parity shared (Sara Chaarawi, 2011)

RAID-6 architecture, double diagonal parity, EvenOdd, allow data recovery in the event of two faults occurring simultaneously in the storage system.

This paper was recommended for publication by Adrian Filipescu

Unfortunately, these methods, once implemented, do not allow scalability of the system; there are implemented for a strict number of hardware, a priori established.

1.1. RAID simple parity

RAID 3,4,5 storage systems, involves creating a simple parity code for data blocks stored on disk arrays, allowing data recovery in case of malfunction. (David A. Patterson, 1998)

The algorithm used to calculate the parity code is:

$$(1)\quad a \oplus b \oplus c \oplus d = p$$

Where a, b, c, d are elements of information (bytes) and p parity code obtained.

Thus, in case of malfunction, recovery of lost informationis made through the parity code, replacing the lost item in above equation by p, and the result is even the missing code. Example:

In case of failure of the element a:

$$(2)\quad a = \oplus b \oplus c \oplus d \oplus p$$

This solution enables easy recovery of data on failure of an information unit. It is easy to implement and scalable, but in case of two simultaneous failure, data are lost.

1.2. RAID 6

RAID 6 requires calculating a second parity code based on Reed-Solomon algorithm, which allows data recovery in case of two simultaneous failures. (David A. Patterson, 1998) (Moon, 2005) (Morelos-Zaragoza, 2006)

Algorithm is of the form:

$$(3)\quad \Sigma c_{i,j} \otimes d_{i,j} = p_i$$

For the particular case where $c_{1,j} = 1$ and the maximum value of j is 4, then we note $d_{1,1}=a$ $d_{1,2}=b$ $d_{1,3}=c$ $d_{1,4}=d$, $p_1=p$ and get :

$$(4)\quad a \oplus b \oplus c \oplus d = p$$

Recovery is slow because of laborious calculations for each disk; the disk array can not exceed a total of 16 disks and is not scalable.

1.3. Hamming Code

Hamming code ECC (David A. Patterson, 1998) (Morelos-Zaragoza, 2006) was used for RAID 2, but because of laborious calculation and low recovery speed is now used only for RAM (ECC).

1.4. EvenOdd algorithm

In 1994, a team from IBM has created EvenOdd algorithm (Mario Blaum, 1995), which allows adding a second parity byte at a simple parity algorithm used by RAID5.

This is an algorithm that provides ease of implementation, speed parity calculation and speed recovery, unlike Reed-Solomon implementation which is slow.

In the following we present EvenOdd coding principle.

Be a set of disks $m+1$, m is in our case equal to 7.

We will use $m\text{-}1$ disks for data storage; disks m and $m+1$will be used for parity storage.

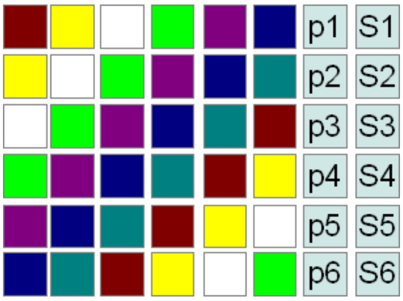


Figure 1 Implementation architecture of the EvenOdd algorithm

Parity of P type is linear parity.

S-type parity is additional parity, calculated as the parity of the inverse of diagonals of the elements $a_{i,j}$ where:

$$(5)\quad Sn = \oplus a_{i,j} \oplus a_{l,t},\ \text{where}\ i \in (j,1),\ j \in (n,i),\ l \in (n+1, m\text{-}2),\ t \in (m\text{-}2, n+1)$$

In the above figure, it can be seen marked with color, diagonals participating in creating Sn parity.

1.5. Diagonal Parity RAID algorithm (RAID-DP)

In 2006 (Chris Lueth, 2010) Network Appliance (NetApp) created RAID-DP algorithm, which is implemented in storage devices it produces. Algorithm, mainly, resembles EvenOdd algorithm, but additional parity calculation is performed on diagonals parallel with the main diagonal, introducing in additional parity calculation, the simple parity calculated along a row.

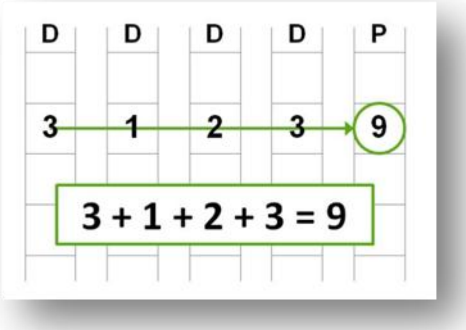
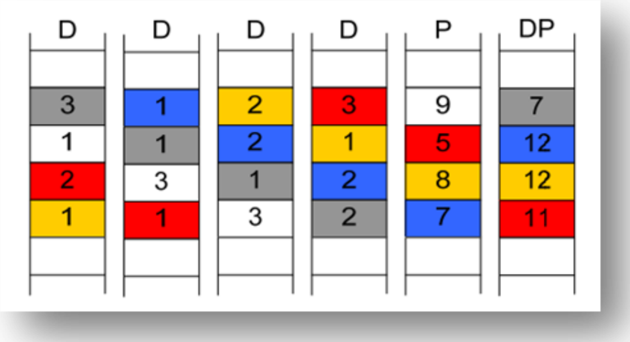


Figure 2 Implementation architecture of the RAID-DP linear parity algorithm

Matrix for parity calculation is ***m-1***. ***m*** is a prime number (in this case 5), and after diagonal parity calculation of ***m-1*** elements, parity is stored in field DP.



D	D	D	D	P	DP
3	1	2	3	9	7
1	1	2	1	5	12
2	3	1	2	8	12
1	1	3	2	7	11

Figure 3 Implementation architecture of the RAID-DP diagonal parity algorithm

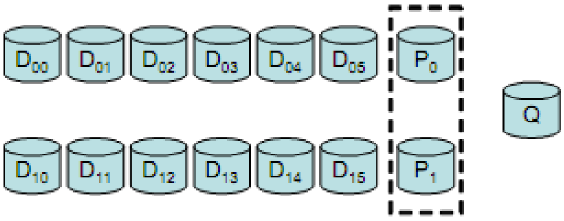
Recovery algorithm takes into account the first line which has two diagonal parity errors. The error is rebuilt with DP. After getting the lost field, the error is recovered from the linear parity P.

1.6. Data storage using common parity

In 2011, Sara Chaarawi and others (Sara Chaarawi, 2011) have published an algorithm that uses only simple parity and allow lost data recovery.

It uses a disk array, which has calculated the correspondent parity for each line. Additional is calculated parity

$$\mathcal{Q} = \mathcal{P}0\oplus \mathcal{P}1$$



If errors occur on any two disks, there can be easily restored in the following conditions:

- not to fail two disks on same RAID line, plus Q disk;
- not to fail three disks from the same line.

2. SOLUTION APPROACH

To store in the Internet, we proposed a system for nodes scalability and increase redundancy for data recovery, while maintaining a reasonable relationship between: storage efficiency, storage capacity and increased speed in data recovery case of failure of the storage system.

Analysis of previous storage methods led to reconsideration of storage methods and achieving data redundancy. Also, inability to increase storage scalability for data retrieval systems that allow data recovery when two failures occurs, led us to consider ad-hoc network topology analysis, created in the Internet.

Internet is created on a mesh type topology, which allows making any particular topology in private networks created on this support.

To store backup data we need a node to perform storage and a backup solution that would solve the access to data if the primary storage node would be unreachable.

A first solution could be a mirror copy of data from node. This would be similar to RAID1.

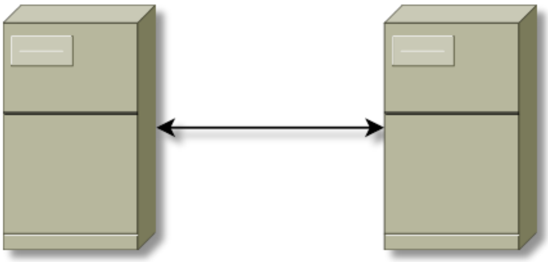


Figure 4. Mirror backup

Such an approach ensures a high recovery rate and if the primary node fails, the secondary node can access the data, being promoted as the primary node. But if both nodes fails, data access is impeded and the data can be lost.

To ensure the data on failure of two nodes, we analyzed computer network topologies that can be performed and algorithms that can be used.

Algorithms RAID 6, RAID-DP, EvenOdd, or common (shared) parity, imply the use of a simple parity algorithm and a second calculation method of an additional parity that allow recovery of stored data. Yet, these algorithms have some inconvenience: increased storage duration, implementation is slow and the system is not scalable.

Providing access to data if a second server fail, has led to an architecture that allows implementation of simple parity for the second server (the mirroring), which permits data recovery, in case of failing of the second server.

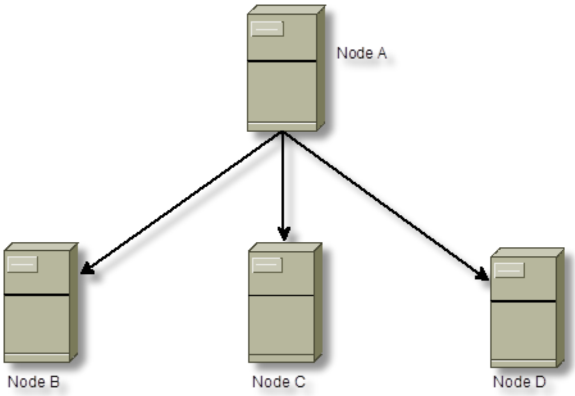


Figure 5 Basic structure

Thus, we come to create a hierarchical topology, which allows data recovery if two nodes fail (with a relatively high recovery rate because the algorithm involved), consisting of a main server and at least 3 servers involved in redundant data storing from the main server. Minimizing the number of servers allows increased data transfer speed between them, and recovery.

Figure 2 shows the basic structure for storing data.

We have a master node, which has 3 slave nodes. Files are stored on the master node A.

Therefore, we can say the following.

Definition. Define as a hierarchical-redundant storage space, a space consisting of a master storage node and at least 3 slave storage nodes, where for $\forall a \in \mathcal{A} \exists b, c, d$ *which satisfies the following conditions (1):*

(6) $a = b + c$

(7) $b \oplus c = d$

where a, b, c, d are elements of stored information.

If b, c, d are bytes then a is the size of 2 bytes.

Meaning for any file A stored in node A, there are three files B, C, and D created from file A, which have size $\frac{A}{2}$ and satisfy the relationship b ⊕ c = d.

At this point we have a storage node A, and 3 replication nodes using simple parity.

Thus, if any two of the four servers fail, data is recovered.

- If nodes **A** and **B** fail, then A data recovery are obtained as follows:

  (8) $a = c \oplus d + c$ for $b = c \oplus d$

- If nodes **A** and **C** fail, then A data recovery are obtained as follows:

  (9) $a = b + b \oplus d$ *and* $c = b \oplus d$

- If nodes **A** and **D** fail, then A data recovery are obtained as follows:

  (10) $a = b + c$ *and* $d = c \oplus d$

We can extend this architecture, and convert each slave node in master node and other nodes remaining slave. Thus we obtain the following architecture (figure 6):

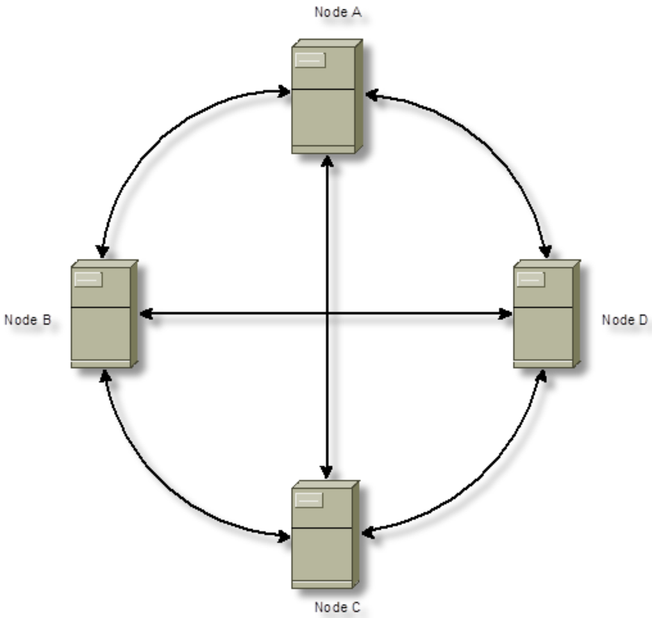


Figure 6 Complete architecture

Currently we have 4 master nodes and 4 slave nodes, with only 4 nodes.

Extending the above argument, we can say that any two nodes may fail, the other two nodes would recover. Actually, it would be an application of the above topology on four nodes, for each one in part.

To refer to this solution, we use as the *storage cell*.

Increasing the number of nodes participating in the creation of a storage cell does not improve in any way the recovering performance of lost data in the cell; the maximum number of nodes that can be recovered is still two.

Increasing the number of errors that can be recovered

In the situation where we have two independent storage cells, can be put in relation with them a third storage cell, which would allow the creation of redundancy between the two storage cells.

Such an architecture is presented below in figure 7.

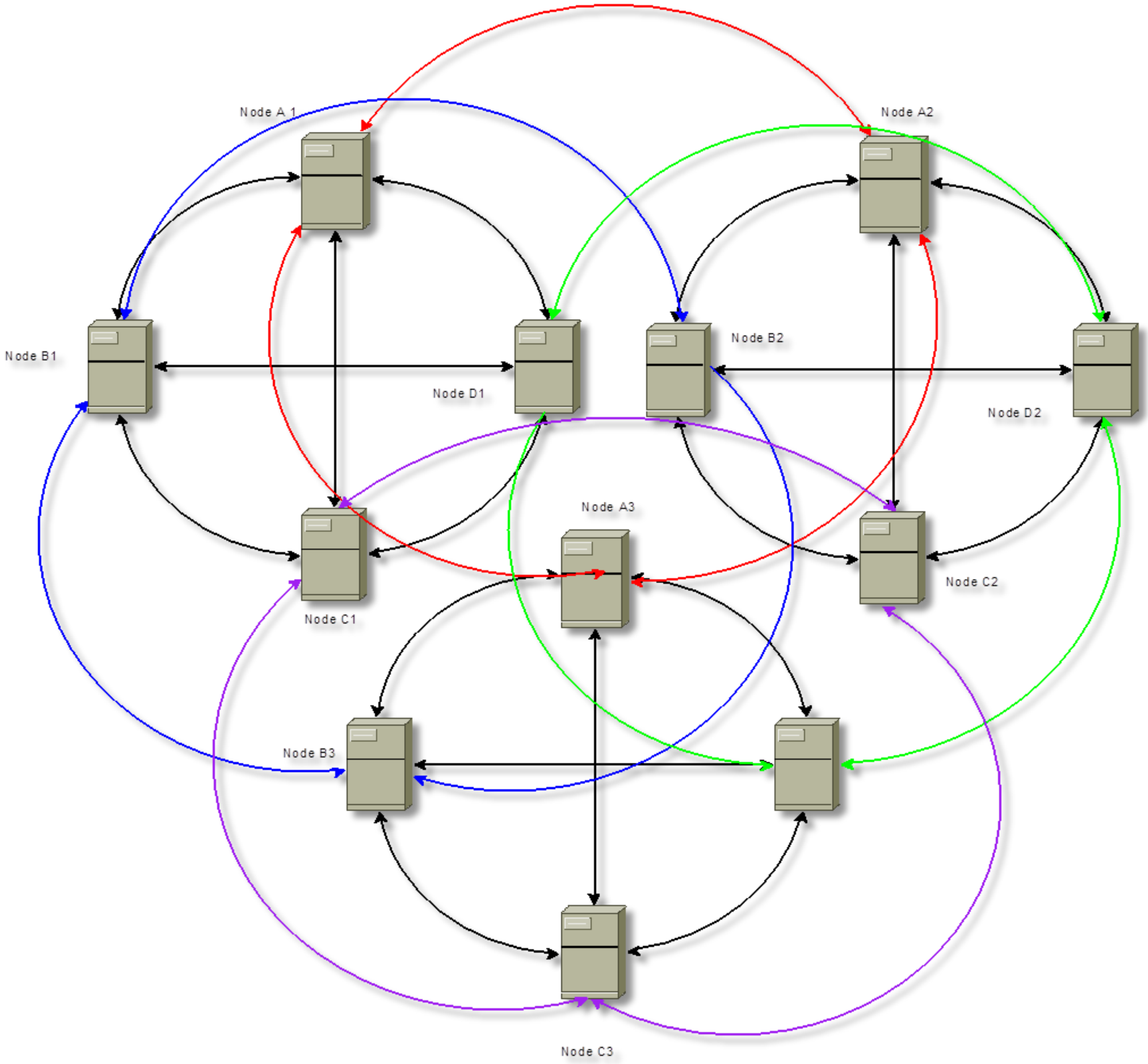


Figure 7. Example of application for 3-cell storage

Equations (2) that define the relationships between nodes in different cells, would be:

(11)• $A'1 \oplus A'2 = A3$

(12)• $B'1 \oplus B'2 = B3$

(13)• $C'1 \oplus C'2 = C3$

(14)• $D'1 \oplus D'2 = D3$

A1, A2, B1, B2, C1, C2, D1, D2 constitutes the information from slave nodes of the master nodes.

From equations (1) and (2) results that every 6 nodes would fall, could be restored from the information stored in the other 6. This leads to the conclusion that if a storage cell is damaged and half of another cell, the data can be recovered in full.

At this point, hierarchical topology first designed for storage, turned into a hybrid topology, by tying in ring the component nodes.

2.1. Storage-recovery algorithm implementation

In order to achieve data-storage as described before, we used a simple parity algorithm. Its implementation was done in different ways to check the speed of storage-recovery.

Storage:

A. Reading source file; dividing into components (files) to be saved on additional storage servers; parity calculation and storing them in parity file.

B. Reading the source file into a memory buffer; parity calculation; write the read data and calculated parity to files.

To restore data, check existing sources, and:

A. If component files exists (resulted from the source file spliting), restoring the source file.

B. If one of the component files missing, is calculated content of the missing file using the remained component file and the parity file.

Implemented algorithms are presented in Annex.

2.2. Storage analysis

To analyze the effectiveness of the proposed solution, we define the following:

Es = maximum storage efficiency (percentage of actual storage capacity required to store initial data of the total storage capacity allocated)

Ps = lost storage capacity, meaning storage capacity lost by implementing redundancy.

Cs = the storage capacity used from the total capacity available.

Ns = storage nodes theoretical necessary from the real storage nodes (implemented).

Name of the maximum efficiency of storage is the ideal case where all data blocks are used. In reality it can not be achieved for various reasons (temporary files, caches, etc).

We believe that we have, in each node, N blocks of memory allocated on a hard drive for data storage.

Storage analysis for basic architecture

Do we assume that the file Fa occupies in node A, M blocks of data. Then files Fb, Fc, Fd occupies each M/2 blocks of data. Therefore, the number of blocks needed to implement redundancy for the file Fa is 2.5*M.

In the ideal case when use all of N blocks for storing in node A, to implement redundancy will result a necessary of more 1.5*N blocks.

$$(15)\ Es=\frac{1xM}{2{,}5xM} \rightarrow Es=0{,}4$$

$$(16)\ Ps=\frac{1{,}5xM}{4xM} \rightarrow Ps=0{,}375$$

$$(17)\ Cs=\frac{2{,}5xM}{4xM} \rightarrow Cs=0{,}625$$

$$(18)\ Ns=\frac{4\ (theoretical\ nodes)}{4\ (real\ nodes)} \rightarrow Ns=1$$

Notice that the redundancy implementation nodes we have unused storage capacity. To optimize storage Cs, should be 1, Ps should aim to 1, to 1 Es should aim to 1 and Ns should aim to higher values.

From the above analysis, result that would be reduced storage capacity that is allocated to implement redundancy. If Node B = Node C = Node D = 0.5 A, then

$$(19)\ Ps=\frac{1{,}5xM}{2{,}5xM} \rightarrow Ps=0{,}6$$

And

$$(20)\ Cs=\frac{2{,}5xM}{2{,}5xM} \rightarrow Cs=1$$

This means judicious use more storage capacity than the previous solution.

Storage analysis for advanced architecture

In this hybrid topology, we have 4 nodes with 4 nodes master slave.

Each master node is slave node to implement redundancy for another master node. This results in a reduction in the number of nodes from 16 to 4.

Suppose we have W storage blocks for each master node. For nodes master to be supported by the slave nodes, it assumes that each node should supplement with 50% of storage capacity of a master node. Thus, for advanced architecture, each node will have at least 2.5*W. In these conditions

$$(21)\ Ns=\frac{16\ (Noduri\ teoretice)}{4\ (Noduri\ reale)} \rightarrow\ Ns=4$$

Advantages of advanced architecture

Advantages of this solution are:

- The possibility of easy data recovery for 4 nodes, where at most two nodes are faulty
- High speed upload and download for file stored in the master node
- Easy scalability of the system, and could easily add a new node, we need only that the equations of data storage nodes to add new slave node.
- By creating nodes distances, increasing the possibility of recovery is obtained for natural disasters, regional.

Analysis of advanced storage architecture

By implementing this solution, r4ise the number of nodes that can fail, allowing the failure of one cell and other two nodes of the other two cells.

$$(22)\ Ns=\frac{32\ (Noduri\ teoretice)}{12\ (Noduri\ reale)} \rightarrow Ns=2,66$$

because we have implemented only two master storage cells and one slave storage cell. If we want to improve the ratio, then we can implement and third master node storage solutions.

3. CONCLUSIONS

Basically, with storage cells, we can implement different storage solutions and data recovery, becoming more efficient, but with specification that will increase allocate storage capacities to implement redundancy.

4. REFERENCES

Chris Lueth, Jay White. 2010. RAID-DP: *NetApp Implementation of Double-Parity RAID for Data Protection*. media.netapp.com. [Online] 2010. [Cited: 05 26, 2011.] http://media.netapp.com/documents/tr-3298.pdf.

David A. Patterson, Garth Gibson, Randy H. Katz. 1998. *A case for redundant arrays of inexpensive disks (RAID)*. Chicago, Ilinois : ACM, Proceedings of the 1988 ACM SIGMOD international conference on Management of data, 1998. pp. 109-116. 10.1145/50202.50214.

Doersken, Trevor. 2008. *Clouds are the user-friendly version of Grids.* SYS-CON : ACM, 2008.

Lueth, Jay White & Chris. *RAID-DP: NetApp Implementation of Double-Parity RAID for Data Protection*. media.netapp.com. [Online] [Cited: 05 26, 2011.] http://media.netapp.com/documents/tr-3298.pdf.

Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. *A Break in the Clouds*. 2009. A Break in the Clouds-Towards a Cloud Definition. 2009.

Mario Blaum, Jim Brady, Jehoshua Bruck, Jai Menon. 1995. EVENODD: *An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures*. IEEE Transaction on Computers. 1995, Vols. 2, pp. 192-202, 44.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski,Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. *A view of cloud computing.* New York, NY, USA : ACM, 2010.

Michael Stonebraker Gerhard A, Schloss. 1990. *Distributed RAID - A New Multiple Copy Algorithm.* Washington : IEEE Computer Society, 1990. ISBN:0-8186-2025-0 .

Moon, Todd K. 2005. *Error Correction Coding.* Hooboken : John Wiley & Sons, Inc., 2005. ISBN 0-471-64800-0.

Morelos-Zaragoza, Robert H. 2006. *The Art of Error Correcting Coding.* Chicehester, West Sussex, England : John Wiley & Sons Ltd., 2006. ISBN-13: 978-0-470-01558-2.

Sara Chaarawi, Jehan-François Pâris,Ahmed Amer, Thomas Schwarz, Darrell D. E. Long. 2011. SSRC: Using a Shared Storage Class Memory

Device to Improve the Reliability of RAID Arrays. *Storage Systems Research Center.* [Online] 2011. [Cited: 09 25, 2011.] http://www.ssrc.ucsc.edu/pub/chaarawi-pdsw10.html.

SSRC: Using a Shared Storage Class Memory Device to Improve the Reliability of RAID Arrays. *Storage Systems Research Center.* [Online] [Cited: 09 25, 2011.] http://www.ssrc.ucsc.edu/pub/chaarawi-pdsw10.html.

Tudoran, Radu. 2011. *Data storage in clouds.* s.l. IRISA,2011. ftp://ftp.irisa.fr/local/caps/DEPOTS/BIBLIO2011/Tudoran_Radu.pdf.