Combining Compression, Encryption and Fault-tolerant Coding for Distributed Storage

Peter Sobe[1] and Kathrin Peter[2]

[1]University of Luebeck
Inst. of Computer Engineering
Ratzeburger Allee 160, D-23538 Luebeck
sobe@iti.uni-luebeck.de

[2]Zuse Institute Berlin
Computer Science Research
Takusstrasse 7, D-14195 Berlin
kathrin.peter@zib.de

Abstract

Storing data in distributed systems aims to offer higher bandwidth and scalability than storing locally. But, a couple of disadvantageous issues must be taken into account such as unreliability caused by faults, temporal downtimes and malicious attacks. To improve dependability, redundancy codes like parity can be used as well as more sophisticated codes such as Reed/Solomon. Another issue - security requirements - arise when data is kept in untrusted units in a network. To encrypt data, it is common to use security algorithms like AES. For efficient transfer and storage, the amount of data can be reduced by compression algorithms. All these techniques - data distribution, fault-tolerant coding, encryption and compression - can be employed together using independent algorithms, but in a proper combination. A superposition of these techniques exploiting synergies is still an issue for research. Thus, in this paper we study proper technique combinations applied to distributed storage. The combinations are classified and examined with respect to their potential benefit and limitations. For our model, performance parameters from the distributed storage system NetRAID are used.

1. Introduction

Distributed data storage potentially allows faster storage access, data durability and a flexible scaling of storage capacity. Many distributed storage architectures appeared, either in closely connected systems such as cluster storage, or in loosely connected environments like the Internet. Storing data in a distributed system involves up to four processing phases for different ambitions. First, a data object is split into blocks and stored across a number of storage units, e.g. disks or storage servers. Splitting data into blocks (striping) allows to assign them to different storage units with parallel access. The second technique, fault-tolerant coding adds structured redundancy that is calculated with respect to the distribution of data, i.e. exploiting independent fault regions. In case that one or more storage units are unavailable, it is possible to recalculate lost data by using this redundancy. As a third technique, compression can be included to reduce the physically needed storage space by removing redundancy from the original data. The access performance may increase because less data has to be transfered. Fourth, data can be encrypted and stored safely outside the users local domain without getting compromised. All techniques cause additional computational effort and the gained benefit depends on several parameters and the requirements in the particular scenario. An example for invoking the addressed techniques is to compress a file (e.g. by gzip), encrypt it in a second step and then store it on a distributed reliable storage system. This sequence is already a good choice to reach low storage overhead and data security for transfer and permanent storage on disks. But it is not necessarily the fastest variant when parallelization on the server side and pipelining of client and server operations is considered. In this paper, a model is presented for deciding about the order of techniques with respect to application constraints, e.g. a secure network-transfer. This model is constructed with knowledge from the implementation of a distributed reliable storage system NetRAID[11, 12]. Parameters are taken from experiments with this system to be practically relevant.

The following sections of this paper are organized as follows. Related work is surveyed in Section 2. Then we continue with the approach to enumerate all possible combinations of the four techniques and preselect them in Section 3, particularly we exclude improper combinations. Then,

1-4244-0910-1/07/$20.00

in Section 4 the alternatives among combinations that are appropriate ones at a first glance are discussed. Combinations with a limited appropriateness are reviewed in Section 5. The model-based comparison of the combinations is presented in Section 6. The results are summed up in Section 7 with implications for distributed storage system design.

2. Related work

Data striping is a common technique, applied either across disks in RAID configurations [6] or across storage servers in distributed systems, e.g. for a media storage system[2] or for parallel storage systems. A variety of such distributed storage systems are available, either as commercial products (IBM-GFS, TerraGrid) or as results from research projects, such as PVFS2[13], Lustre[3], NetRAID[12], ClusterRAID[15] and OceanStore[7, 9]. Most of them are considered for closely connected systems within a trusted environment and thus do not include encryption. The exception is OceanStore, a real wide-area storage system among the examples given.

Data distribution is a more general principle that includes striping but also irregular distribution and replication. Systems like Freenet [4] aim to share data in a distributed decentralized system using peer-to-peer techniques. In these systems, encryption techniques are applied in order to forbid untrusted peers to get insight into data and also to release them from the responsibility for the data stored for other peers.

Compression is mostly an unconsidered feature for distributed storage. One may explain this situation by the common practice to compress data prior storage, e.g. applying data specific compressors – image and video codes for instance. Compression often is integrated into an application and many data formats already represent data in a compressed form. But, compression is a technique that can also be placed at other system components, e.g. in the network layer or in the storage system (compressing file systems, e.g. ZFS[8]).

The combination of encryption and compression is addressed in [14] and in [5] especially for text data. In [14] the two steps are combined to reduce processing time by adding a pseudo-random shuffle into the data compression process. In [5] the approach is to transform text data into an intermediate form which can be compressed more efficient. They apply a strategy called Intelligent Dictionary-based Encoding for preprocessing and encryption.

For channel coding, the common practice is to prepare messages for transmission by (1) encryption and then (2) employ coding for error tolerance. Normally, compression is placed before encryption, because encryption transforms its input into a sequence that is hard to compress in a subsequent step. A trustworthy encryption algorithm should generate output data that can not be well compressed [10].

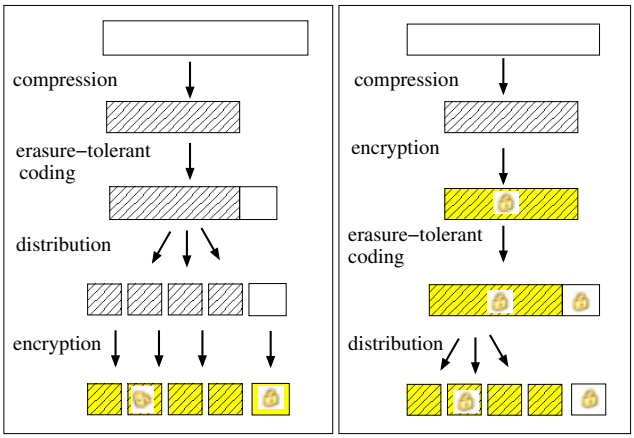

Figure 1. Examples of combinations

A system that allows to use encryption, compression and distribution is the Storage Resource Broker (SRB) [1]. SRB is a distributed Grid storage system that creates a global view over a number of heterogeneous storages. For higher security a client has the option to encrypt and compress the data object locally and then to store it in the SRB space. For fault-tolerance the data object can be replicated, that is managed with the help of the SRB meta-data catalog MCAT. SRB differs from our model (and as well from NetRAID) regarding the impossibility of data striping that is a key feature in NetRAID. Encryption also is a feature of a few file systems, e.g. EncFS for LINUX or the encrypting file system extension for NTFS/Windows.

3. Preselection of Combinations

When combining distribution, fault-tolerant coding, compression and encryption, the steps can be arranged in several orders. With a number of 4!=24 different possibilities, more or less appropriate operation sequences can be formed for distributed storage. Two possible combinations are illustrated in Figure 1.

For a preliminary selection we group the techniques in all possible orders. Combinations that are obviously not appropriate shall be identified in a first step. The appropriateness is constrained by a couple of aspects:

- Deletion-tolerant codes rely on independent fault regions. Bits/blocks in a codeword are assumed to be stored on different storage resources and thus, distribution has to be done after deletion-tolerant coding.
- Compression prior distribution across several storage units introduces interdependencies among distributed data. Thus, compression must not be placed between fault-tolerant coding and distribution.

- Fault-tolerant encoding should not be directly followed by encryption. If encryption contains cipher block chaining, it introduces dependencies among data that later on will be stored in different fault regions.

A limited application of encryption or compression after fault-tolerant encoding is still possible when the scope of the operations is limited to blocks within a fault region.

In order to obtain a compact notation, a letter symbol for each individual technique is used. A group of four letters is used to characterize a particular order, e.g. *CDEF*. The letters are chosen as following:

D ... Data distribution *F* ... Fault-tolerant encoding
E ... Encryption *C* ... Compression

According this notation, in Table 1 all 24 combinations are enumerated, together with a note if they are appropriate, useful in a limited way or an improper choice.

Summarizing, 6 combinations are valid ones without any limitation (marked with Yes in Table 1). Further 6 combinations are in a limited way useful. The limitation is that compression or encryption have to take the fault-tolerant coding into account and may not work across blocks assigned to different fault regions. The rest, further 12 combinations are not practical (marked with No). In the next sections we describe the valid combinations more detailed. Particularly, the computation time and the storage overhead is specified.

4. Appropriate Combinations

Six combinations are proper ones, with a sound order of the four techniques. After deletion-tolerant encoding, data is split and stored in parallel on a number of storage units. The distribution pattern follows the separation into different fault-regions. This circumstance is expressed by the continuous subsequence *FD* which is present for all appropriate combinations. Each combination is getting evaluated in terms of time required for storing the data and the resulting storage overhead. The time is decomposed into the following components:

T_D ... Time for distribution, includes data transfer over the network
T_F ... Time for fault-tolerant encoding
T_E ... Time for encryption
T_C ... Time for compression
T_S ... Time for storage access, always as the last step

Further, a few parameters define the concrete system, e.g. the number of storage resources used for data striping.

d ... number data blocks (striping factor)

no.	order	appropriate	reason
1	CDEF	No	F after D does not exploit indep. fault regions (d1)
2	CDFE	No	(d1) + E after F does not support decryption after faults (d2)
3	DCEF	No	see (d1)
4	DCFE	No	see (d1) and (d2)
5	EFCD	Limited	compression between fault-tolerant coding and distribution destroys structured redundancy (d3)
6	EFDC	Yes	Compression less effective but parallel
7	FECD	Limited	see (d3)
8	FEDC	Limited	Encryption possibly across regions that need to be independent for structured redundancy (d4)
9	CEDF	No	see (d1)
10	CEFD	Yes	
11	ECDF	No	see (d1)
12	ECFD	Yes	Previous encryption causes sub-optimal compression (d5)
13	FDEC	Yes	(d5) + compression less effective
14	FDCE	Yes	
15	DFEC	No	see (d1)
16	DFCE	No	see (d1)
17	CFED	Limited	see (d4)
18	CFDE	Yes	
19	FCED	Limited	see (d3)
20	FCDE	Limited	see (d3)
21	EDCF	No	see (d1)
22	EDFC	No	see (d1)
23	DECF	No	see (d1)
24	DEFC	No	see (d1)

Table 1. Combination of techniques in orders

k ... number redundant blocks
n ... blocks after fault-tolerant coding; $n = d + k$
c ... compression factor, reciprocal of the reduction factor, i.e. $c = 5$ when the compressed size is a fifth of original size, $size_{compr} = \frac{1}{c} size_{original}$
e ... encryption loss, reduction of compression factor, $e \in \{\frac{1}{c} \dots 1\}$, $e = 1$ when no reduction is observed, and $e = \frac{1}{c}$ when reduction reaches a maximum effect
s ... separation loss, reduction of compression factor due to compression on separate blocks $s \in \{\frac{1}{c} \dots 1\}$, $s = 1$ when no reduction is observed and $s = \frac{1}{c}$ when reduction reaches a maximum effect

The combined influence of encryption and separation to the compression efficiency is bounded to a maximum value that totally compensates the compression effect, thus $e \cdot s \geq \frac{1}{c}$.

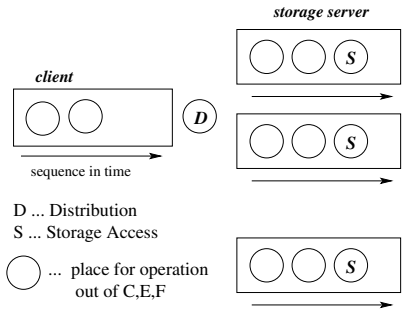

Figure 2. System model

This combined loss l is specified as follows:

$$l = \begin{cases} e \cdot s & \text{if } e \cdot s \geq \frac{1}{c} \\ \frac{1}{c} & \text{otherwise} \end{cases}$$

The computational effort for fault-tolerant encoding linearly grows with the number of additional storage resources for redundancy information. Thus it depends on k. In contrast, this computational effort is independent from d, the number of storage resources used for the data. This can be explained by the amount of original data which has to pass the coding algorithm. This amount does not change when the striping factor changes. Another observation is the size of the redundancy information that is - seen from a single redundancy storage - equal to the size of the data on a particular data storage, the $\frac{1}{d}$-fold of the original file size. The system model is shown in Figure 2, illustrating the access by a single client to a farm of parallel storage servers. Operations can be assigned to the client or to servers, solely (i) distribution is always placed at the interface between client and servers and (ii) storage access is always that last operation in the sequence (thus not included in the combinations). In the following, equations for the time of a write operation and for the storage overhead will be given. These equations are used in a model-based evaluation of the operation sequences in Section 6.

EFDC - Encryption-FTCoding-Distribution-Compression
Data is encrypted at first and then redundancy for fault-tolerant storage is added. Data blocks are distributed across a number of storages and there each data block is compressed separately. An advantage of this combination is that data is secured at client side at the beginning.

$$\begin{aligned} T_{EFDC} &= T_E + k \cdot T_F + \frac{n}{d}T_D + \frac{1}{d}T_C + \frac{1}{d \cdot c \cdot l}T_S \\ O_{EFDC} &= \frac{1}{c \cdot l} \cdot \frac{n}{d} \end{aligned}$$

The space efficiency can not be optimal - as described above. Due to prior encryption and the separation into shares on different storage units, compression is supposed to be less efficient.

CEFD - Compression-Encryption-FTCoding-Distribution
Compression is applied at first and the size of the data object will be reduced. Then the subsequently executed encryption and fault-tolerant coding work on a reduced data size and thus will be faster. At the end data is distributed across the storage nodes. The compression of the whole data object and its placement before encryption leads to an efficient solution.

$$\begin{aligned} T_{CEFD} &= T_C + \frac{1}{c}T_E + \frac{k}{c}T_F + \frac{n}{d \cdot c}T_D + \frac{1}{d \cdot c}T_S \\ O_{CEFD} &= \frac{1}{c} \cdot \frac{n}{d} \end{aligned}$$

Beside the time efficiency, this variant is also space-efficient with the best compression rate that solely depends on the input data.

ECFD - Encryption-Compression-FTCoding-Distribution
Data is encrypted and compressed on the client side and then stored on a fault-tolerant storage system. This is a valid but a less efficient scheme. Encryption usually increases the entropy of the data, so that compression will be less efficient and more data must be handled by the deletion-tolerant encoder.

$$\begin{aligned} T_{ECFD} &= T_E + T_C + \frac{k}{c \cdot e}T_F + \frac{1}{d \cdot c \cdot e}(nT_D + T_S) \\ O_{ECFD} &= \frac{1}{c \cdot e} \cdot \frac{n}{d} \end{aligned}$$

FDEC - FTCoding-Distribution-Encryption-Compression
After fault-tolerant coding data is split and stored in parallel on a number of disks. Each data block is encrypted and then compressed. The compression efficiency is lower because entropy is typically increased with encryption. Additionally, compression on small blocks is another factor for a less efficient compression. The advantage of this variant is the parallel compression and encoding. The disadvantage is the plain-text transfer of data to the storage resources. Any sequence DxE, with $x \in$ {F,C,FC,CF,_} indicates such a plain text data transfer.

$$\begin{aligned} T_{FDEC} &= k \cdot T_F + \frac{n}{d}T_D + \frac{1}{d}T_E + \frac{1}{d}T_C + \frac{1}{d \cdot c \cdot l}T_S \\ O_{FDEC} &= \frac{1}{c \cdot l} \cdot \frac{n}{d} \end{aligned}$$

FDCE - FTCoding-Distribution-Compression-Encryption
This combination is similar to FDEC, with the difference that data is compressed at first and than encrypted. The advantage is a better compression factor.

The plain text transfer of data is still a point of criticism.

$$T_{FDCE} = k \cdot T_F + \frac{n}{d} T_D + \frac{1}{d} T_C + \frac{1}{d \cdot c \cdot s}(T_E + T_S)$$
$$O_{FDCE} = \frac{1}{c \cdot s} \cdot \frac{n}{d}$$

CFDE - Compression-FTCoding-Distribution-Encryption
Data is encrypted after distribution, in parallel for each data block. With the compression step at first, this variant is space-efficient, comparable with the CEFD combination. A disadvantage can be seen in the plain text data transfer - as encryption is done only at the storage units.

$$T_{CFDE} = T_C + \frac{k}{c} \cdot T_F + \frac{n}{d \cdot c} T_D + \frac{1}{d \cdot c}(T_E + T_S)$$
$$O_{CFDE} = \frac{1}{c} \cdot \frac{n}{d}$$

5. Combinations with Limited Appropriatness

Further six combinations are useful with the limitation that at least one operation - compression or/and encryption - is to apply separated onto several blocks. The block boundaries were introduced by a previous erasure-tolerant coding that assigns blocks to independent fault-regions.

EFCD - Encryption-FTCoding-Compression-Distribution
Data is encrypted and then redundancy is added for fault-tolerance. Original data and redundant data is compressed and distributed. To preserve the fault-tolerance property it has to be ensured that compression is not applied across the blocks.

$$T_{EFCD} = T_E + k \cdot T_F + \frac{n}{d} T_C + \frac{1}{d \cdot c \cdot l}(n\, T_D + T_S)$$
$$O_{EFDC} = \frac{1}{c \cdot l} \frac{n}{d}$$

When there is no other reason for applying FT-coding before compression, one should first compress, then encrypt and apply the erasure-tolerant code at last. This leads to CEFD, which is appropriate without limitation.

FECD - FTCoding-Encryption-Compression-Distribution
After adding redundancy, data is encrypted and compressed. Encryption and compression has to take into account that the structure of the redundancy is not violated, by working within the block boundaries. In a last step, data is distributed block-wise.

$$T_{FECD} = k \cdot T_F + \frac{n}{d} T_E + \frac{n}{d} T_C + \frac{1}{d \cdot c \cdot l}(n\, T_D + T_S)$$
$$O_{FECD} = \frac{1}{c \cdot l} \frac{n}{d}$$

FEDC - FTCoding-Encryption-Distribution-Compression
After adding redundancy for erasure-tolerance, the resulting file is encrypted, again with the requirement that the structured redundancy is kept. After distribution data blocks are compressed separately.

$$T_{FEDC} = k \cdot T_F + \frac{n}{d} T_E + \frac{n}{d} T_D + \frac{1}{d} T_C + \frac{1}{d \cdot c \cdot l} T_S$$
$$O_{FEDC} = \frac{1}{c \cdot l} \frac{n}{d}$$

CFED - Compression-FTCoding-Encryption-Distribution
Data is at first compressed to reduce the data size. In a next step, structured redundancy for fault-tolerance is added and data gets encrypted. The encryption is to apply on separate blocks.

$$T_{CFED} = T_C + \frac{k}{c} T_F + \frac{n}{d \cdot c} T_E + \frac{n}{d \cdot c} T_D + \frac{1}{d \cdot c} T_S$$
$$O_{CFED} = \frac{1}{c} \cdot \frac{n}{d}$$

FCED - FTCoding-Compression-Encryption-Distribution
After erasure-tolerant encoding, the code blocks are compressed and encrypted separately for blocks. Then the blocks are distributed across the storage units.

$$T_{FCED} = k \cdot T_F + \frac{n}{d} T_C + \frac{1}{d \cdot c \cdot s}(n\, T_E + n\, T_D + T_S)$$
$$O_{FCED} = \frac{1}{c \cdot s} \cdot \frac{n}{d}$$

FCDE - FTCoding-Compression-Distribution-Encryption
FCDE is similar to FCED, but encryption is shifted at the last position and executed at the side of the storage servers. Encryption can be done parallel on several blocks which should result in a performance improvement. This improvement must be traded for the disadvantage of plain–text data transfer over the network.

$$T_{FCDE} = k \cdot T_F + \frac{n}{d} T_C + \frac{1}{d \cdot c \cdot s}(n\, T_D + T_E + T_S)$$
$$O_{FCDE} = \frac{1}{c \cdot s} \cdot \frac{n}{d}$$

6. Experimental Evaluation

In this section, first realistic parameters are discovered and then applied on the model to compare the combinations, particularly the write operation of data to the distributed storage system. The parameters c, e, s will be determined by experiments on different data. For that, we applied the

operations compression, encryption and distribution to several files. The experiments follow the paths shown in Figure 3. The obtained values are listed in Table 2. Three classes can be identified – well compressible ($c \approx 5.4$), moderately compressible ($1.4 < c < 2.6$) and practically non-compressible ($c \approx 1$).

The time factors in the formulas T_E, T_C and T_S are assessed by observing data processing rates for the files and shown in table 3. For a further analysis, data rates are converted into processing times for a normalized file size of 1 GByte.

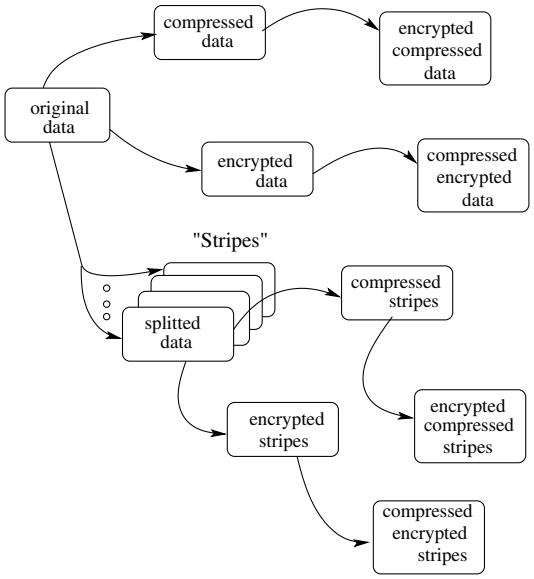

Figure 3. Evaluation paths

file	size (Byte)	compress. factor (c)	loss factors e	s
channel_0	1024000	5.45	0.183	0.95
cfiles.tar	512000	5.42	0.1854	0.9196
scifi-book.pdf	2024743	1.446	0.691	0.9897
lecture.pdf	1076754	2.54	0.3929	0.9923
j7jac.mat	6233334	1.004	0.996	0.9996
bluemoon.mp3	23711411	1.0082	0.992	0.9999

Table 2. Observed parameters on several file types

Particularly T_D, the distribution time that includes network transfer and T_S that is strongly dependent on the particular storage resources are taken from measurements on the NetRAID system. Thus, to provide realistic parameters for T_F, T_D and T_S, measurements with different functional coverage were performed. For one experiment, the functionality covered solely the erasure-tolerant coding at the client side, whereby the other functions were deactivated in the NetRAID system. With that, the rate for erasure-tolerant coding could be measured. Another measurement setup included erasure-tolerant coding and distribution of data across the storage servers. In that case, the actual access to the server disks was deactivated and the rate for the two steps - erasure tolerant-coding and distribution - could be measured. In a third step, we measured the rate for the complete storage system activity, including access to the server disks. Two storage layouts were selected for further analysis Parity(7+1) and Reed/Solomon(6+2). The operation times (normalized to a file size of 1 GByte) and the storage overhead are plotted in Figure 5 for a parity code and in Figure 6 for a R/S code.

operation	rate (MByte/s)	normalized Time (s)
Encrypt (AES)	19.9	$T_E = 50.25$
Compress (gzip -best)	15 . . . 50	$T_C = 20 \ldots 66,\overline{6}$
Disk write	30 . . . 50	$T_S = 20 \ldots 33.\overline{3}$

Table 3. Observed rates for encryption, compression and disk access

The rates are converted to operation times on a file size of 1 GByte. In result, the time consumption for erasure-tolerant coding (FTCoding), distribution and disk access on servers can be extracted. These time values are used for evaluation by formulas presented in Section 4 and 5, with factors in order to express the ratio of actual transferred data size and file size - as assumed in the formulas. The particular times and factors are depicted in Figure 4.

In Figure 5 the time consumption of a write operation and the storage overhead are plotted for a storage system with a parity code. The plot's x-axis classifies the three regions of compression factors. Figure 6 depicts the modelling results when an R/S code is applied. The results show that wrt. operation times three classes originate. The first class are combinations that are fast (low operation times). A slight positive influence of the compression effect is visible. Members of this class are FDEC, CFDE, FCDE and FDCE - with FDCE as the fastest combination. All combinations in this class execute encryption on the server side in parallel and thus gain operation speed.

A second class are combinations that are slower, but still faster than many others when data can be compressed well. Members of this class are FEDC, CFED and CEFD. Combinations with an encryption prior distribution can be found in this class - for secured network transfer. The rest of combinations - a third class - are slow wrt. operation speed. The reason is that compression has no effect due to prior encryption and in most cases parallelization of the time consuming

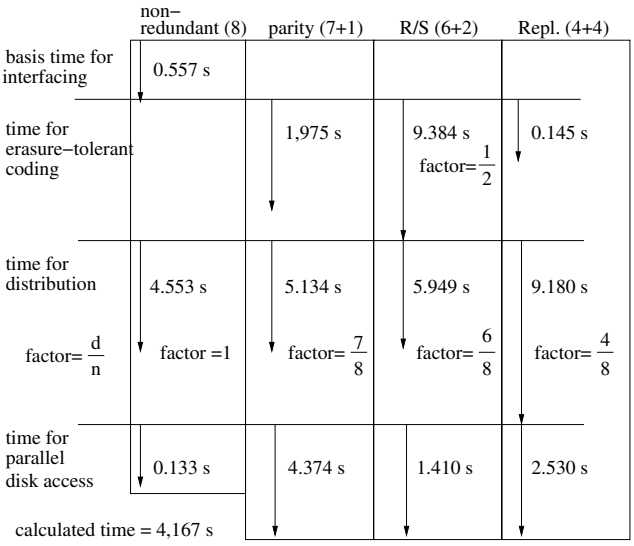

Figure 4. Times from experiments with factors for application in model

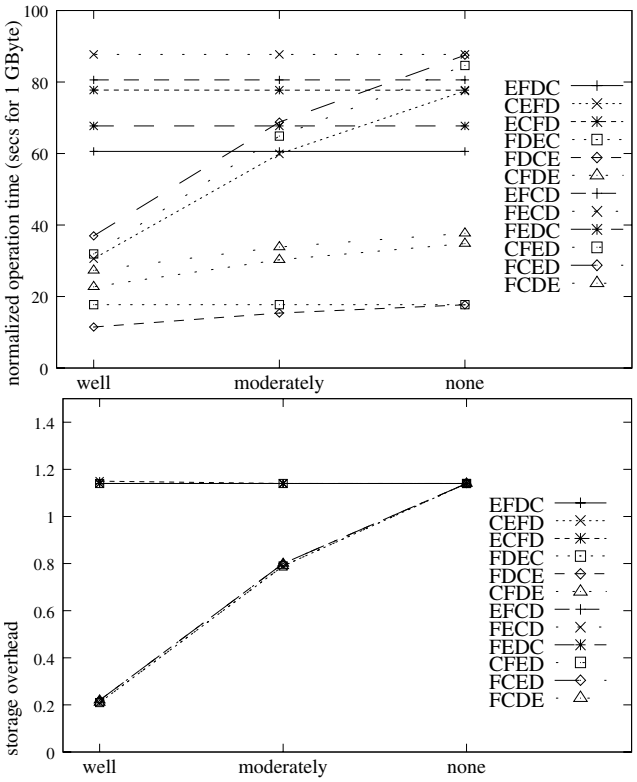

Figure 5. Comparison of combinations with parity(7+1) as erasure-tolerant code.

parts is not applied.
Regarding overhead, only two classes originate, a class with combinations that can exploit the reduction of data size for efficient operation and the second class that does not. Members of the first class (the advantageous ones) are: CEFD, FDCE, CFDE, CFED, FCED and FCDE. It is obvious that solely the fact that compression is placed prior encryption is the reason for reaching the low overhead.

7. Implications for Storage System Design

From our measurements and the modeling, a couple of design rules for distributed storage systems can be reasoned:
Encryption and Compression – Encryption typically increases the entropy of the symbol stream. So it is disadvantageous to apply compression after encryption. The only way is to use the techniques in a combined algorithm, examples for such combinations can be found in [5, 14]. If there is an encryption on client side, compression should be placed before encryption. Another alternative is to move both compression and encryption to the server nodes. This allows efficient parallelization of the most time-consuming operations. It provides safety against attacks on storage resources, and against theft of storage nodes. It can protect privacy for data stored on untrusted Internet peers as long as the key for encryption can not be used again for decryption.

Erasure-tolerant Coding and Compression – These techniques are opposed because coding adds structured redundancy while compression removes any kind of redundancy. If data is compressed before coding to decrease data space, the amount of original data as input for redundancy coding is reduced. This is combined with an encoding time reduction. Compression still can be done separately on parts of data that is already assigned a single fault region. It was shown that separation does not strongly influence the compression efficiency. So, compression on server nodes is a fast variant, when data does not have to transferred securely and the network bandwidth is not a limiting factor.

Erasure-tolerant Coding and Encryption – Encryption can be applied prior erasure-tolerant coding without any limitation. The erasure-tolerant codes work on any kind of data streams, independently if plain data or encrypted data is processed. To apply encryption after erasure-tolerant coding introduces the restriction that no dependencies across data on several fault regions may be introduced, for instance by cipher block chaining. So, the scope of encryption must be restricted to separate regions on data. As encryption is relatively time consuming - in spite of efficient algorithms - one may place encryption on the server side.

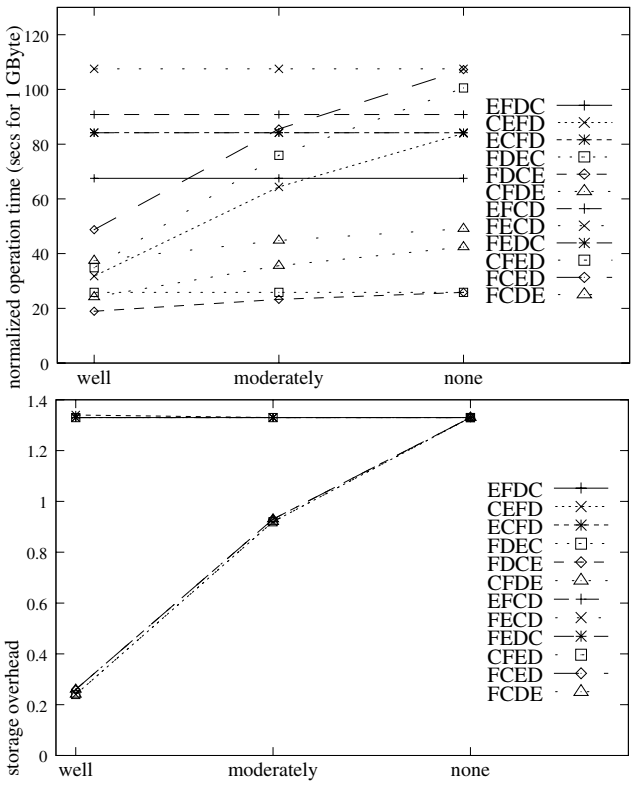

Figure 6. Comparison of combinations with R/S as erasure-tolerant code.

8. Conclusions

Appropriate combinations of compression, encryption, erasure-tolerant coding were identified for distributed storage systems. We modeled the chain of operations, taking into account the compression effect, loss factors by the influence of encryption and data separation. Parallel processing on the server side allows to move time-consuming operations, e.g. encryption, on the server side. The model parameters are taken from measurements on a real distributed storage system, to be realistic. For example, the sequence compression, encryption, deletion-tolerant coding, distribution and storage access reaches the least storage overhead, is relatively fast and provides secure network transfer. Faster systems can be build by shifting compression and encryption to the server side for parallel operation. Using the model, one is able to select a appropriate combination either with the focus on a combination that provides the highest access bandwidth or the lowest storage overhead. Constraints in the operation environment, e.g. a low-bandwidth network can be considered in the model with according parameters. A future objective is to let the storage system components decide autonomously about the appropriate combination, taking user and system constraints into account.

References

[1] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press, 1998.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. Technical Report CSD-930042, University of California, Computer Science Deptartment, December 1993.

[3] P. J. Braam et al. *The Lustre Storage Architecture*. 2004. Cluster File Systems Inc., http://www.lustre.org/docs/lustre.pdf.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46ff, 2001.

[5] V. Govindan and B. Shajee-Mohan. An Intelligent Text Data Encryption and Compression for High Speed and Secure Data Transmission over Internet. In *International Conference on Information Technology Coding and Compression, ITCC*. IEEE Computer Society, April 2005.

[6] R. Katz, G. Gibson, and D. Patterson. Disk System Architectures for High Performance Computing. In *Proceedings of the IEEE*, pages 1842–1858. IEEE Computer Society, Dec. 1989.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[8] S. Microsystems. ZFS: the last word in file systems - http://www.sun.com/2004-0914/feature/. 2004.

[9] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and K. J. Maintenance Free Global Data Storage. *IEEE Internet Computing*, pages 40–49, September/October 2001.

[10] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[11] P. Sobe. Data Consistent Up- and Downstreaming in a Distributed Storage System. In *Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os*, pages 19–26. IEEE Computer Society, 2003.

[12] P. Sobe and K. Peter. Construction of OR-based Deletion-tolerant Coding Schemes. In *IPDPS 2006 Proceedings, Workshop on Dependable Parallel, Distributed and Network-centric Computing*. IEEE Computer Society, 2006.

[13] P. D. Team. Parallel Virtual File System, Version 2. http://www.pvfs.org/pvfs2/pvfs2-guide.html, 2003.

[14] C.-E. Wang. Cryptography in Data Compression. *CodeBreakers Journal, Security and Anti-Security - Attack and Defense*, 1, 2006.

[15] A. Wiebalck, P. Breuer, V. Lindenstruth, and T. Steinbeck. Fault-Tolerant Distributed Mass Storage for LHC Computing. In *CCGrid 2003*, 2003.