Publication I

Mikko Pitkänen, Rim Moussa, Martin Swany, and Tapio Niemi. 2006. Erasure codes for increasing the availability of grid data storage. In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006). Guadeloupe. 19-25 February 2006. Pages 185-194. ISBN 0-7695-2522-9.

Erasure Codes for Increasing the Availability of Grid Data Storage

Mikko Pitkanen
Helsinki Institute of Physics
Technology Programme
mikko.pitkanen@hip.fi

Rim Moussa
ISSAT Universite 7 Nov de Carhage
Department of Computer Science
rim.moussa@issatm.rnu.tn

Martin Swany
University of Delaware
Dept. of Computer and Info. Sciences
swany@udel.edu

Tapio Niemi
Helsinki Institute of Physics
Technology Programme
tapio.niemi@cern.ch

Abstract

In this paper, we describe the design of a highly-available Grid data storage system. Increased availability is ensured by data redundancy and file striping. Redundant data is computed using Reed-Solomon (RS) codes. The level of availability can be chosen for each individual file. Storage overhead is minimal compared to traditional redundancy strategies based on complete replication. Our prototype uses existing Grid data management tools (GridFTP) for communication, and the RS encoding and decoding is embedded in the client software. Performance measurements in wide area network prove the efficiency of our design for high-availability and striped file transfers.

1 Introduction

A traditional mechanism to secure against data loss is to replicate, i.e. mirror, the data. In replication, multiple verbatim copies of the file are created. These copies can then be distributed to different locations in the network. With replication entire files are used and the distribution, access and management of data is straightforward. However, the storage overhead with replication is always higher than it is with erasure codes. When a certain level of availability is targeted the erasure codes are able to provide service with a lower storage overhead than replication techniques.

One of the well known erasure coding systems with low overhead that has been widely deployed is the one used in RAID [13] disk arrays. Despite the obvious advantages, software-basd deployments of similar schemes have been missing thus far. However, different approaches to use erasure codes for high redundancy in storage applications have been recently discussed in the literature. Some of the work has shown how erasure codes can be used to implement scalable data storage in local cluster environments [11]. The other studies have suggested the use of erasure codes for file storages in wide area environments [17, 6]. However, implementations for real world use and their wide scale deployment is still something that has not been seen. In this paper we will present an implementation of a file system client that uses erasure codes and striping to ensure high availability of files in Grid environment.

The Grid community has developed a data management infrastructure that can be used to share and access data and files securely in a distributed environment. We envision that our application can benefit from features available in the existing Grid and can provide value without requiring changes to components that are finally beginning to become standards. In particular, we benefit from the single-sign-on framework as our application, by definition, communicates with several secure systems. Moreover, the GridFTP [9] protocol provides a good solution for data transfer in our scheme. The mere requirement of digital certificates and service oriented architecture makes Grid environments a more stable platform to employ erasure codes than more frequently changing peer-to-peer networks with high churn.

We employ erasure coding techniques for same purposes that they are used in well known RAID disk arrays. However, in loosely coupled environment the capability to survive from one or two simultaneous failure is not usually considered to be enough. As a solution we present a storage system where the level of redundancy can be chosen freely to meet the characteristics of the environment. Moreover, we will present performance figures of our implementation to enable the feasibility assessment of proposed design.

The structure of this work is as follows. In Section 2

Proceedings of the Advanced International Conference on Telecommunications and
International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)
0-7695-2522-9/06 $20.00 © 2006 IEEE

we will present the related work and give an overview on erasure coding. In Section 3 the structure of our implementation is explained. In Section 4 we provide performance measurements both for the local coding operations and for the involved data transfer operations in Grid environment. In Section 5 we conclude our work and discuss future directions.

2 Related Work

Our proposition is to apply erasure codes for data storage in a Grid environement using GridFTP. To the best of our knowledge there is no such earlier work. Here we briefly review relevant work done within the erasure code community. Additionally, we refer to related work that has been done to study the distributed storages and parallel file access in the Grid community.

2.1 Erasure Codes

In our work we use Reed Solomon codes based on Vandermonde matrices. The code that we use is based on the work by Rizzo [18], which can be referred for a good explanation of the working of the code. The RS erasure codes can be used to encode k data items into $n > k$ encoded data items. Later, it is sufficient to have any k out of existing n items to decode the original information. In our scheme this means that the original data can be recovered even if $n - k$ of the stored data items become unavailable. In addition, the used code is a systematic code meaning that the original k stripes form a verbatim copy of the input file. If these stripes are available the decoding is unnecessary and the file needs to be only reassembled.

Low-Density Parity-Check (LDPC) codes provide an alternative for performing erasure coding to achieve high redundancy with low storage requirements. The characteristics of these codes are different than those of Reed Solomon codes. Codes of the LDPC family can perform very fast coding but they require slightly higher storage overhead. However, a recent work by Collins and Plank [14] shows how the RS codes can have better performance, regardless of the rate of the encoding, when used with relatively small n. We suggest use of coding in Grid environments where even a modest amount of redundancy can be seen to provide significant value. This is due to the assumption that in certificate based service infrastructure the churn is lower and the entities more trustable than in unstructured P2P environments. Thus, we think that our choice to use RS codes is justified.

From this point forward the discussion merely focuses on comparing the characteristics of erasure codes and replication when they are used for file distribution. Replication, or mirroring, is a traditional mechanism used for file distribution in the Internet content distribution, peer-to-peer file sharing [7] and in Grid infrastructures [16, 2].

2.2 Transferring Files in Wide Area Environment

Replication is very simple to implement and it provides fast access to data. Moreover, it provides redundancy and load balancing to file sharing environments. The requirement to facilitate fast access to frequently used data has made replication one of the central techniques in Grid data management.

PAST is a peer-to-peer storage utility which replicates files around the network[7]. In PAST the efficiency of file access is achieved by using a routing scheme that finds replicas closest to the user. In addition,, PAST uses caching of popular files to improve efficiency. Another distributed storage utility called OceanStore [17] uses similar overlay routing architecture to find nearby replicas. The OceanStore also discusses the use of RS erasure codes for the archival layer of their data storage. They show how the erasure codes can be used to provide significantly higher availability figures than replication with similar space requirements.

In the Grid community a replication framework is developed to establish a data management infrastructure where high availability, fault tolerance and minimal access times are targeted. To minimize data access latencies automatic replication and replica selection frameworks have been developed. By taking into account the networking monitoring data during automatic replica selection the access time in wide area transfers can be significantly reduced [10]. Other work on Grid data management has studied the coupling between data access and the scheduling of jobs. A study by Ranganathan and Foster [16] showed that it is efficient to schedule works close to data, but the data replication can be an independently coordinated process. A study by Cameron et al.[3] shows that is important to consider both the data access and computing node queues in order to optimize usage of computing and storage resources. Hence, even though we stripe the data in our work, we see that it is important to maintain locality information of data to enable the data access with minimum costs.

When a file is moved in a network using our scheme the operation involves transferring many fractions of the file. This causes the load imposed by the transfers to be more distributed across network links and servers. In this respect we are very likely to see similar phenomena in our system to those observed in a study by Vazhkudai [20] where the data sets were accessed by using different parts of data sets located in different network locations. In that study, co-allocated transfers were used to access multiple Grid sites simultaneously in order to download parts of the accessed data set. The co-allocated downloads were seen to perform

better than accessing the best replica of the data set, especially when the files were large.

2.3 Using Erasure Codes to Archive Data

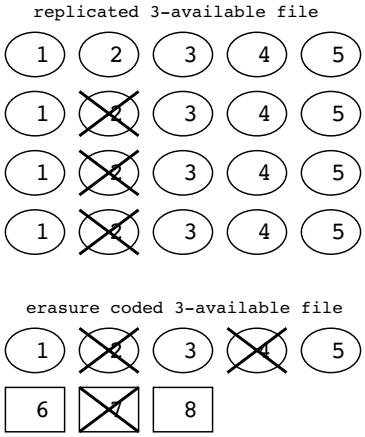

Figure 1. 3-available replicated file and 3-available RS encoded file (k=5,n=8)

Figure 1 illustrates the storage space requirements in different file distribution schemes. In our example the file has been stored in such a way that any $(n - k) = 3$ missing fractions can be tolerated; values $n = 8$ and $k = 5$ are used as an example. With replication, multiple copies of the entire file (consisting of parts 1 through 5) are stored into separate places. The division of file into stripes in the replication picture illustrates the approach used in BitTorrent [5]. BitTorrent divides the file into pieces which are then replicated and distributed in P2P network. Fractions of a file from different sources can be used to reconstruct the file. The BitTorrent approach provides load balancing in the network but still has similar storage space requirements as traditional replication when high availability is targeted.

Buyers and Luby [1] have suggested a solution called Digital Fountain where Tornado codes are used for many-to-many file transfers. The codes they use have properties of the LDPC family of codes. In their solution the storage overhead is slightly larger than with RS codes, but very fast coding is achieved. The Tornado code solution is feasible for setups where many clients simultaneously access data from many sites. Another study by Collins and Plank [6] point out features of LDPC codes that make them less suitable choice for our storage design. Not All sets of different stripes in LDPC codes can be used to perform decoding. Thus the metadata management and server scheduling for LDPC codes are more complicated when it cannot be assumed that any n different stripes are good for reconstruction.

We propose a design for an archival storage where less frequently used files are stored so we are not currently aiming to benefit from Tornado code based solution's ability to serve large user populations with larger overhead. Instead, we think that the approach presented in the OceanStore work [17] would be suitable for our design. The suggestion is that files once decoded from the archive can be made available in faster accessible layer of storage. According to this evaluation the downloading party may make a once decoded file available through replica management infrastructure.

3 Storage Application

In this section we present our implementation for a Grid data storage client which uses erasure codes. We explain how we have implemented all the required functionality as a client application. Due to this, the proposed implementation does not require any changes to existing Grid data management infrastructure. All novel functionality that we propose takes place in client computer and only files need to be processed by the Grid infrastructure. We see this as an advantage since the file management services are one of the most stable parts of existing Grid infrastructure.

3.1 Client Architecture

The functionality of proposed storage application is implemented as a Java client application than can be flexibly run in various platforms. The modular design of the application has helped us to reuse existing software, as we benefit from two external modules. For erasure coding operations we use a sightly modified version of an open source implementation for RS erasure codes [12] on Java. For the Grid functionality we use an open source Java Commodity Grid (CoG) Toolkit [4].

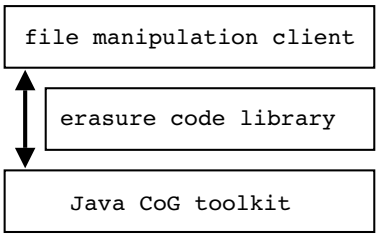

Figure 2. Architecture of storage client implementation

Figure 2 illustrates the layered structure of our client application. In the topmost level we have developed a software layer that coordinates working with files and stripes of files during the encoding and decoding processes. In the encoding phase the topmost layer reads packets of the input file

and passes these to the erasure coding library. The erasure coding library then produces redundancy packets for the input information and the packets are returned to the file manipulation client, which writes redundant information to n files. In addition to that a file distribution description is created to contain the parameters of encoding. In the decoding phase, the functionality is the opposite, the file manipulation client reads blocks from at least k files to reconstruct the original file.

After the encoding phase, the data is distributed to storage elements around the network. The client application uses GridFTP functionality of the CoG toolkit to securely transfer the stripes of data to their destinations. After the stripes are transferred, the information about their location is added to a distribution description. The description contains in a file the information needed to locate the stripes and the appropriate parameters for reconstructing the original copy from the striped data. In the beginning of the file retrieval, the application uses GridFTP client to download the stripes according to the description. After at least k stripes have arrived, the erasure coding library is used to reconstruct the file under the control of the file manipulation client.

3.2 Distribution Descriptions

One cost of using erasure codes is that during the decoding the identity of the blocks needs to be available to be able to reconstruct the original data. However, this information usually has negligible size compared to the amount of the stored data. To manage the required book-keeping task we have created a distribution description format that contains the needed information to retrieve files from our storage.

```
#FEC Properties for : test.file
#Thu Aug 25 15:29:28 CEST 2005
k=5
n=9
packetSize=20000
fileLength=104857600
test.file.0.tmp=db-ibm.fri.utc.sk
test.file.1.tmp=pchip69.cern.ch
test.file.2.tmp=pc15.hip.fi
test.file.3.tmp=stout.pc.cis.udel.edu
test.file.4.tmp=n2grid.pri.univie.ac.at
test.file.5.tmp=moonshine.pc.cis.udel.edu
test.file.6.tmp=pchip11.hip.fi
test.file.7.tmp=db-ibm.fri.utc.sk
test.file.8.tmp=pchip69.cern.ch
OPTION: CHECKSUM INFORMATION
```

Figure 3. Description for erasure code based file distribution

Figure 3 illustrates a distribution description containing the information about a striped and distributed file. The stored parameters enable retrieval of stripes and define parameters used in decoding the original file. In the figure the arguments k and n are striping factors as explained earlier. Attribute *packetSize* describes the size of used encoding packet and the self-explanatory *fileLength* is stored as bytes. The mappings from a file stripe to a storage element name contain the contact information to each hosting server. Moreover, we have extended implementation to store checksum information for ensuring integrity of the stored data.

3.3 Security Considerations

An important security implication that our solution has is the increased level of information privacy. When using replication for high availability many verbatim copies of the file are distributed to the network. Thus, any single system in the network may become compromised revealing the entire contents of the stored file. When using erasure codes to stripe files, however, a malevolent party needs to gain access at least into k systems in order receive the entire contents of a file. In addition to gaining access to these k systems the attacker needs to get the file description to recover the original file.

The initial design that we present in this paper stores the file distribution meta data in the client computer. Only fractions of the stored data are distributed to GridFTP servers, which are identified by digital certificates. As we are dealing with ordinary files, these can also be encrypted with well known algorithms if a higher level of privacy is required for stored information. Moreover, when only a part of a file is transferred to each destination the possibility for an eavesdropper to gain the entire file on the way is significantly smaller.

Work by Rabin [15] mentions a possible attack against a distributed erasure coded storage. In this attack a stripe of encoded data would be replaced with a different data D. When the data D is used in the decoding process it would produce a faulty file. Similar to Rabin's suggestion to use fingerprints to secure against such scenarios we have implemented the possibility to store checksum information in the file descriptions. However, we do not see that the need for checksum information is more important than with replicated content. The security implications and solutions to them will be further addressed in our future work.

3.4 Network Wide Effects

When we propose our design to be used in a wide area environment we need to evaluate how our design effects the overall system operation. We store the data in compact form but in addition to storage space requirements we also want

to minimize the amount of data moved in network. When a $(n-k)-available$ file is stored to network the replication approach requires transferring and storing data $(n-k+1)*filesize$. With the erasure code approach that we propose the amount of data moved and stored is significantly lower, namely $(n/k)*filesize$.

During the download each server is loaded by requesting $(1/k)$ fraction of the file, instead of the entire file. Thus, we achieve load balancing on the servers. Moreover, traffic in the "first mile" and in the wide area connections is divided between multiple links and network segments. The "last mile" connection in the client end will be, however, a link that all the pieces of the file have to pass. For a good evaluation on related transfer issues refer to [20].

The code that we use has a desirable property in that the level of availability can be freely adjusted to meet the requirements of the environment. In a Grid environment we have relatively high level of trust in resources and we can benefit from relatively small redundancy, e.g. parameters in the range of $k = 5$ and $n = 8$, which can survive from three simultaneously unavailable stripes of data. When large values for n are needed, e.g. in environments with high churn, other codes of the LDPC family have shown better performance [14]. The additional coding work our design imposes on the system takes place in the client computer where the owner of the resource can easily prepare for heavy tasks.

4 Performance Evaluation

In this section we present our results from performance measurements that we have conducted using our client implementation. The measurements are run on AMD Athlon 3000 computer with 1 GB memory and 300 GB hard disk. We used Java 1.42 running on the Fedora Core 3 Linux distribution. For each measurement, test files containing arbitrary data were generated. First, we will discuss the general perspective of encoding performance and discuss the related issues. Second, we will explain issues that are faced when manipulating files and provide related performance measurements. Finally, we focus on the effect of network performance in our distribution scheme.

4.1 Erasure Coding Performance

We have conducted performance measures with a real implementation of our client application to evaluate feasibility of using software erasure codes for file distribution in wide area environment. Some earlier work [18, 14] provides measurements results for performing erasure coding of data with real software implementations. However, these measurements provide performance figures which are achieved by running erasure coding and decoding operations for data held in main memory. Thus, the performance figures cannot be generalized to a general purpose file manipulation client.

We present a general design for file storage that is able to work with file sizes bigger than the amount of free memory. Thus, we have decided to use an approach where the data is always stored to disk between operations that code the file block-by-block. The performance measurements that we have conducted show that the processing overhead caused by file manipulations is not negligible. Especially, we face large amount of input/output (I/O) operations. However, with the chosen approach we are able to process very large files and build a storage that is broadly applicable.

We considered two different approaches to interleave erasure coding and networking operations. In our approach we perform erasure coding operations in one step and all network operations in a separate step. Another approach would have been to transfer stripes over network while doing the encoding, as has been proposed in Digital Fountain solution [1]. By choosing the latter approach we would have avoided the local disk bottleneck and achieved higher level of parallelism. However, with chosen approach the controlling of transfers is easy to manage and we avoid complicated working in the event of connection breakdowns. Moreover, we need to transfer only entire files over the Grid infrastructure and do not require any changes to the existing Grid service framework.

	MATHEMATICAL	I/O
ENCODING	57%	40%
DECODING	83%	14%

Table 1. Erasure coding performance profile, k=5 and n=8.

Table 1 summarizes the execution profile of the storage client during the local file striping and reassembling phases. 97% of the most significant tasks were divided into mathematical operations related to erasure coding and to I/O operations related to local file manipulations. The statistics were collected by running the client application on Java virtual machine with option `-Xprofile`. For the above figures attributes ($packetSize = 20kB$, $k = 5$ and $n = 8$), were used and random files of 1GB were processed a total of 5 times to get the average results shown in the table. In the following discussion we use *original stripes* to refer to stripes that contain the information in the original file. The label *Parity stripes* is used to refer to stripes that have been constructed by the erasure code based on the original stripes. Before measuring the decoding performance three of the original stripes were removed to recover them through the use of erasure codes. With a systematic erasure code the file can be reassembled from the original stripes. When all original stripes were present the decoding simply

consisted of I/O operations.

The computational complexity of the erasure codes is usually seen as the main issue related to their applicability. However, when manipulating large files we can notice how the I/O operations also play significant role in the overall performance. To enable working with files larger than main memory we must expose to frequent disk-to-memory-to-disk operations. Hereafter, *coding performance* is used to include the mathematical coding and related local I/O operations in our storage scheme.

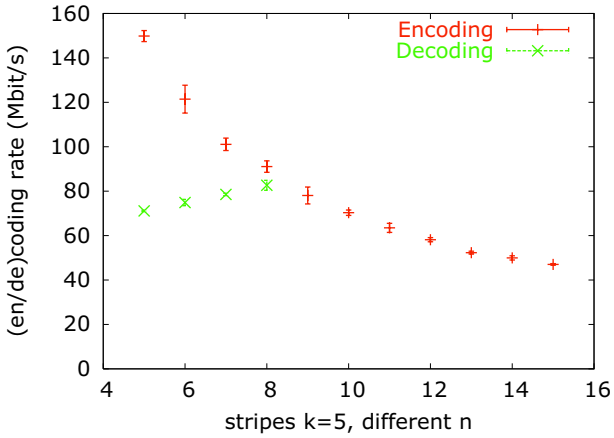

Figure 4. Coding performance with $k = 5$ and different n.

Figure 4 illustrates erasure coding performance in a measurement where a 1 GB file was coded from $k = 5$ to different levels of redundancy. With encoding the measurements are straightforward; at each step file is striped to $k = 5$ pieces, which are then encoded to $n = 5...15$ stripes. With decoding the results are harder to illustrate as the the initial setting may be any of the numerous alternatives. Some of the original stripes as well as some of the parity stripes may be missing, the only requirement being that at least k stripes must be present. In the figure we have plotted decoding performance for $n = 8$ so that the $x-axis$ indicates how many of the n stripes were present. At each step one of the original files are removed and we can observe how the decoding is faster when more original data is available. In the encoding performance we can see decreasing performance as n increases. The decrease is due to increased amount of work producing n units of data in the encoding phase and larger number of I/O operations caused by writing to more output files. As an example we observe the case where $n = 8$ and the encoding speed is approximately 90 Mbit/s. Here we have decided to use bits per second for throughput measures since we are working in a network environment. We will later see how this performance relates to throughput experienced in wide area file transfer operations.

4.2 Manipulating Files With Erasure Codes

In our scheme we are interested in running erasure codes and file operations at the same time. These two classes of operations have significantly different requirements from the system. The erasure codes are computationally intensive requiring CPU processing, while the file manipulations require heavily on I/O performance.

In the encoding procedure we read the file through block by block. Every block consists of k packets each having size $packetSize$ and these blocks are encoded in a way that they expand to n packets. After encoding, the packets are written to n files. After writing, the next block of the original file is taken to processing. In the decoding phase we proceed in an opposite way, at least k files are read, each for a different stripe within the same block and the packets from each stripe are decoded to form the original file block by block.

Performance of erasure codes for encoding and decoding scales linearly with the packet size for the coding. The longer the packets are the fewer the number of needed packets. So the overall time spent doing the coding for a file remains the same. However, in our application the packet size has an effect on the amount of the I/O operations. The larger packet size results in less frequent need to read data from disk to main memory. We can thus optimize the performance of the application by choosing an appropriate packet size for the I/O operations.

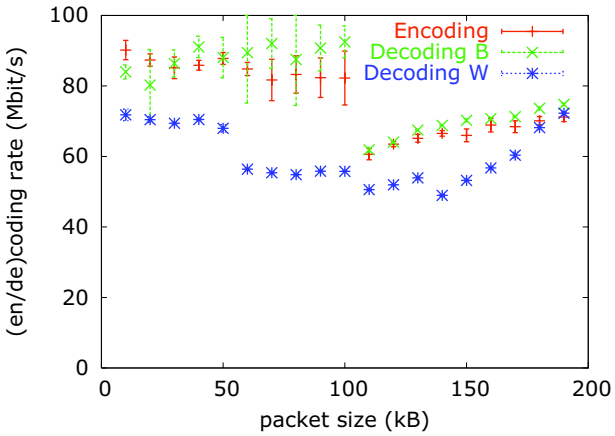

Figure 5. Effect of packet size on coding performance.

Figure 5 shows the effect of packet size on the coding performance. In the measurement example parameters $k = 5$ and $n = 8$ were used with a file size 1 GB. We have plotted three cases. For encoding we simply encode $k = 5$ packets to $n = 8$ packets. For decoding we plotted the best case, where we have all original files available to reassem-

ble the file. The worst case figures for decoding illustrate a case where maximum number $n - k$ of the original stripes were missing and maximum number of reconstruction takes place. Other initial setups for decoding are expected to have performance that falls between these extreme cases . Figure 5 illustrates that the chosen packet size actually has a considerable effect on the coding performance.

During coding a large amount of data needs to be passed through the memory hierarchy consisting of hard disk, main memory and CPU cache. Thus, the optimal packet size is likely to show dependency on the system configuration. We do not want to provide optimal value for the packet size but rather encourage the user to configure the parameter on system-by-system basis. In our measurements packet sizes from 10 kB to 50 kB were seen to perform well. Future work will include a test harness to automatically determine the best values for a particular configuration.

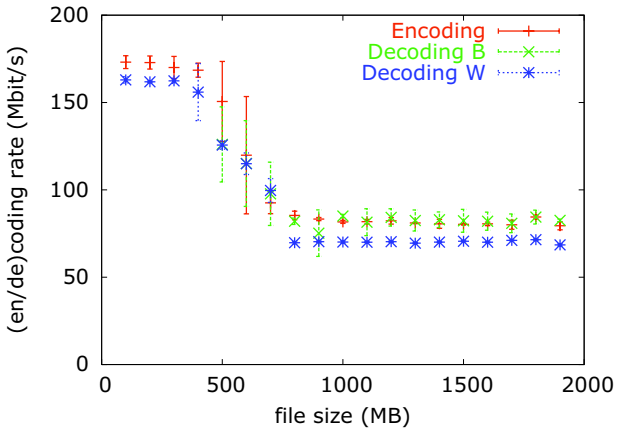

Figure 6. Effect of file size on coding performance.

Figure 6 shows the effect of file size on the coding performance. In the measurement parameters $k = 5$ and $n = 8$ were used with a packet size of 20 kB. Again, we have used the maximum number of missing original stripes to measure the worst case decoding performance. The best case for decoding performance has been measured by reassembling the file from the original stripes. The figure shows how performance can be significantly better with small files which easily fit into the main memory. With large files, the need to perform frequent transfers between disk and memory becomes the performance bottleneck. We use the performance with 1 GB file as our reference performance as the performance does not seem to drastically decrease with larger files.

As we are aiming for a general solution we must consider performance values that are achieved with files beyond the capacity of available main memory. However, the significantly faster coding with small file sizes can be benefitted from in a scenario where a third party provides service for file recovery and storage retrieval. The best case decoding performance for small files between 100 MB and 300 MB are not seen in figure5 since the figures were around 500 Mbit/s. That fast decoding performance can be achieved when the original stripes need only to be reassembled in main memory.

In this section we have shown that the structure of the client system is a significant factor when applying erasure codes for a file storage. As our goal is a general purpose design where large files may be stored and archived for high redundancy, we must consider performance factors beyond optimization of the erasure coding itself. With careful choice of parameters, performance can often be significantly improved.

4.3 Transferring Striped Files in Network

In this section we show how the time needed to perform the coding for a file relates to the time needed to transfer the stripes of the files over a wide area network. To perform our measurements, we use the client to distribute $n = 8$ stripes to four different storage elements. For the following measurement we have established a testbed consisting of four storage elements in different geographical locations and a client computer. The components of the testbed are located as follows: the client and one of the storage elements reside at CERN, one storage element is in Slovakia and two storage elements are in Finland. All the components of the testbed are connected to a high speed academic network to enable a experiment setup that is characteristic to data intensive Grid infrastructures.

We have used three file sizes to study how the size of the manipulated file affects the relationship between the coding overhead and the network transfer times. We repeated measurements 40 times with file sizes 10 MB and 100 MB and 10 times with larger 1000 MB file. The average performance of these transfers has been plotted for all the figures. For each stripe the client application creates a separate thread to control the transfer between each pair of endpoints. The authentication and connection establishment step is carried out separeately for each storage connection.

Figure 7 shows the required work during the process of uploading a file with proposed storage design. First, the file is encoded from $k = 5$ to $n = 8$ stripes and this is illustrated as the lowest portion of each bar. Second, the time for Grid security operations including authentication and authorization are recorded for each of $n = 8$ transferred stripes, the average value is plotted as the second lowest part of each bar. Third, the time to complete all n transfers is plotted in the upmost part of the bars. Thus the two upmost portions of each bar illustrate the overhead spent in Grid

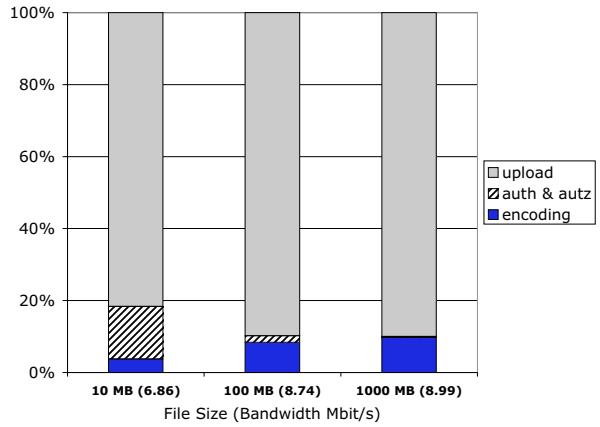

Figure 7. Coding delays compared to network delays during file upload.

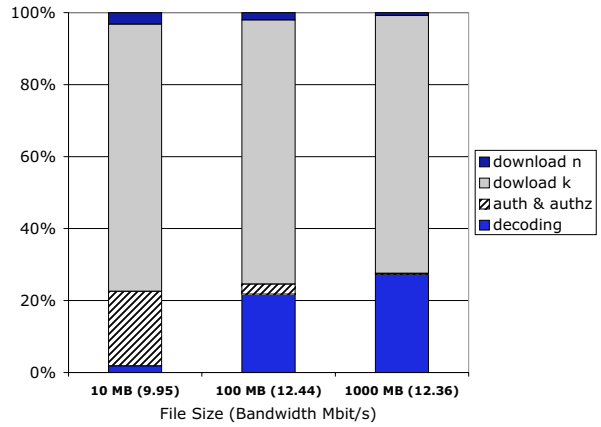

Figure 8. Coding delays compared to network delays during file download.

security and wide area network operations.

The upload measurements illustrate how the time used to encode a file in the client application is relatively small compared to network related delays. With small file sizes, the encoding takes even less time than the time used to authenticate the user to storage resources. The proportional time used for encoding increases with the larger files, but as we have shown earlier in Figure 6, the coding performance is close to constant with files larger than 1000 MB. We can see that the originator of the file does not seem to experience processing overheads higher than 10% during the file archiving procedure. The amount of data transferred is $(n/k) * filesize$ and the client computer and its network connection obviously experience an increased amount of traffic. However, in the server end the network and the server need to process only $(1/k) * filesize$ of data and efficient load balancing can be achieved in the wide area environment. Moreover, compared to fault tolerant replication, less storage resources have to be allocated from each server.

Figure 8 illustrates the required processing overhead when a file is accessed by our storage application. The lowest portion of each bar describes the decoding processing where the file is reassembled. The second lowest part of the bar illustrates the time used by Grid security operations. The third part illustrates the time to complete the first $k = 5$ stripes. This is the earliest time the decoding can begin. The upmost part of the bars is the time to complete the $n - k$ remaining transfers. The decoding could be started just after the arrival of k:th stripe. However better decoding performance can be achieved when more stripes are available as was illustrated in figure 4.

The download figures show again how the time to recover the original file from the stripes is significantly lower than the time needed to transfer the data over the network. The proportional time to decode the original file increases with the file size but remains in the range of 25% of the total time with files of 1000 MB. In case of slow connections our client can stop the slow transfers and files from these connections can be treated as unavailable stripes and recovered through decoding. The download procedure has similar load distribution effects in the network as the upload phase. Aggregated traffic of all stripes is experienced in the client end and the access cost is balanced between several servers and network links in the wide area.

The measurements that we have conducted are dependent on the network performance between the client computer and the storage element sites. However, we have chosen to use storage sites that are have features characteristic to an academic Grid environment. Many repetitions were used to come up with describing averages and figures can be considered to be representative. In future research we plan to address the distribution model of transfer and coding performance and the effect of data transfer scheduling on the performance distributions.

4.4 Comparison Against Replication

In the following we will provide measurement results that compare the performance of our design to file replication approach. We compare the striped transfer to parallel transfer of multiple replicas, the comparison is based on the observation that in both cases the file will be available even if $n-k$ connections cannot be formed or if any $n-k$ storage elements are unavailable.

Figure 9 compares a striped 4-available upload to an

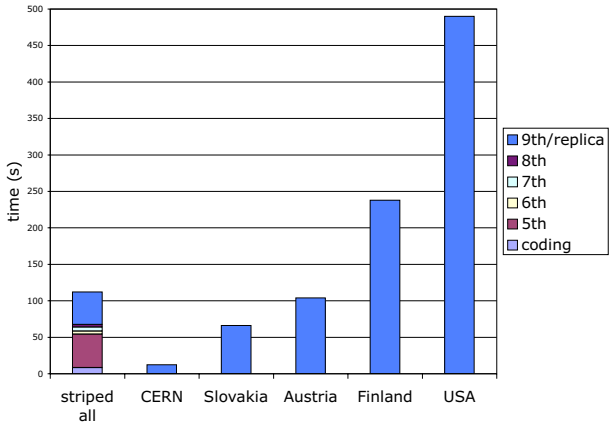

Figure 9. Global 4-available file upload compared to replication.

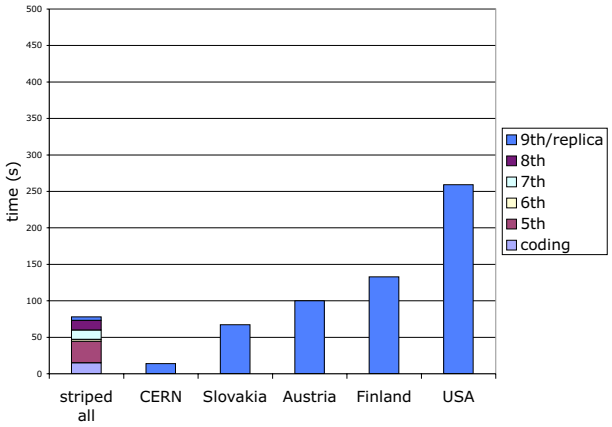

Figure 10. Global 4-available file download compared to replication.

equally available replicated upload. For striping we have used parameters $k = 5$ and $n = 9$ and for replication 5 replicas are concurrently uploaded. The used file size is 100 MB and the results are an average of 10 measurements. In the experiment we have used seven GridFTP servers in five geographically distributed sites. The storages are located in Switzerland, Austria, Slovakia, Finland and the USA and the client application was run in Switzerland. It can be seen how the striped file upload time is significantly lower than the replica transfer to costly locations. Even when two stripes are transferred to a site in the USA and two to Finland the overall upload time stays much lower than transferring entire file to a costly location. Moreover, the amount of data that needs to be transfered for 4-availability in case of striped data is $(9/5) * filesize$ against $5 * filesize$ with replication approach.

Figure 10 compares striped 4-available download to a equally available replicated download. The measurement setting is the same as in the experiment from Figure 9. Again the striped download shows good performance when compared to parallel downloads of replicas. The arrival time for each stripe is plotted and it should be noted that the decoding can be started immediately after the first k stripes have arrived. The decoding performance will then depend on how many of the original stripes are available for reconstruction. We can observe in both upload and download measurements that our solution loses in performance comparison against access to a nearby replica. However, good performance can be sustained even when some of the data is accessed through slow connections or loaded severs.

5 Conclusions and Future Directions

In this work we have proposed a novel solution for providing fault tolerant Grid data storage. The solution we propose is able to provide high level of redundancy and load balancing with minimum storage overhead. The cost of using our storage application is essentially the coding overhead in the client application. The advantage of using erasure codes lies in significantly lower storage space requirements and balancing the load across multiple servers. We have shown with a real world implementation and realistic testbed that it is feasible to use erasure codes for data storages in the Grid environment.

With an example case with k=5 and n=8 we we experienced 10% overheads during the upload and 25% overheads during the download caused by the erasure coding. At the same time 3-availability of the files were achieved with only 40% of the storage space requirements as opposed to replication approach. Clearly the justification to choose our approach to store a file is dependent on the expected time to store, and the frequency of file access. If a very fast and frequent access to files are required a replication infrastructure [10] can be used in addition to our application.

We have proposed a design where the coding operations are always performed in the client computer using entire files and stripes of files. The chosen approach enables robust operation and easy management of transfers in the event of connection breakdowns. However, a tradeoff with the chosen approach is that we temporarily need $(n/k) * filesize$ amount of free disk space to store the file stripes between coding and network operations. This space would not be needed when sending encoded information immediately to network servers. We will continue our

work to study how coding time and transfer operations can be coordinated in the Grid environment in an optimal way.

5.1 Future Work

Work done in OceanStore project [17] discusses how the level of availability can be maintained in an erasure code based file archive. The suggestion is that the owner of data would sweep over the fragments of a file and if the availability has dropped below a threshold value, a repair process would take place. In the repair process the owner of the file would recalculate and redistribute the files to the storage. We plan to extend our work to study how the Grid data management infrastructure would be used to perform the same task in an optimal way. Especially, we envision services such as replica location service [2] and file transfer service [8] to be used for these operations.

Wide scale deployment of erasure codes in Grid environment will change the load patterns in wide area networks. In particular, load balancing between servers is an evident result. As the procedure of file access has significantly different characteristics, the optimal file placement and scheduling strategies as studied in [16, 3], are also likely to change. To study the effect of the wide scale deployment of our design, we plan to extend our studies the PlanetLab environment (please see `http://www.planet-lab.org` for more information).

Recent work in erasure codes has given motivation to study new classes of codes to be applied to network file storages. Whereas the RS codes have been shown to be suitable for our solution some other codes may be more suitable when applied in different manner. In particular we are interested to study the applicability of LDPC [14], Raptor [19] and LT [19] codes.

Acknowledgments

We wish to thank people who have helped us to establish the testbed used in the experiments. The good community spirit in the Grid community has made the work inspiring. This work was partly supported by the Finnish Academy of Science grant no 205961.

References

[1] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *INFOCOM*, pages 275–283, New York, NY, Mar. 1999. IEEE.

[2] M. Cai, A. Chervenak, and M. Frank. A peer-to-peer replica location service based on a distributed hash table. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 56, Washington, DC, USA, 2004. IEEE Computer Society.

[3] D. Cameron, A. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini. Analysis of scheduling and replica optimisation strategies for data grids using optorsim. *Journal of Grid Computing*, 2(1):57–69, March 2004.

[4] Java Commodity Grid (cog) Kit. http://www.cogkit.org/, referenced November 30, 2005.

[5] B. Cohen. Incentives build robustness in bittorrent, 2003.

[6] R. L. Collins and J. S. Plank. Assessing the performance of erasure codes in the wide-area. In *DSN-05: International Conference on Dependable Systems and Networks*, Yokohama, Japan, 2005. IEEE.

[7] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of HotOS VIII*, 2001.

[8] File Transfer Service. `http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/transfer/index.htm`, referenced November 30, 2005.

[9] GridFTP: Protocol Extensions to FTP for the Grid. `http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf`, referenced November 30, 2005.

[10] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. File-Based Replica Management. *Future Generation Computer Systems*, 21(1):115–123, January 2005.

[11] W. Litwin, R. Moussa, and T. J. Schwarz. LH*$_{RS}$ - a highly-available scalable distributed data structure. *Journal of ACM TODS*, 35, 2005.

[12] Open source FEC library for Java. `http://www.onionnetworks.com/developers/`, referenced November 30, 2005.

[13] 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*, pages 109–116, New York, NY, USA, 1988. ACM Press.

[14] J. S. Plank and M. G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *DSN-2004: The International Conference on Dependable Systems and Networks*. IEEE, June 2004.

[15] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[16] K. Ranganathan and I. T. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *J. Grid Comput.*, 1(1):53–62, 2003.

[17] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage, 2001.

[18] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *SIGCOMM Comput. Commun. Rev.*, 27(2):24–36, 1997.

[19] A. Shokrollahi. Raptor codes. Technical report, Laboratoire d'algorithmique, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2003. Available from `http://algo.epfl.ch/`.

[20] S. Vazhkudai. Enabling the co-allocation of grid data transfers. In *nternational Workshop on Grid Computing GRID*, pages 44–51, 2003.