A formal analysis of recovery in a preservational data grid

Niels H. Christensen
Royal Library of Denmark, Dept. of Digital Preservation
& Netarkivet.dk
nhc@kb.dk

Abstract

A data grid made for the long-term preservation of digital materials is described. The data grid's ability to recover from data loss is analysed by developing a formal, mathematical model for the relevant, implemented software operations.

1. Introduction

This extended abstract[1] presents an analysis of a system built for long term data preservation. The analysed system was developed by Netarkivet, the national Danish web archive, and is used for storing large amounts of crawled web pages. The system is in essence a data grid based on distributed, online file systems. The analysis investigates its capabilities for recovering from events that cause damage to or loss of data.

1.1. Motivation and related work

The threat of latent failures. The recent paper [1] presents a detailed list of threats to the long-time preservation of digital materials, and discusses some of the dangerous assumptions digital preservationists may fall prey to. One important aspect studied in the paper is fault visibility. Some failures (e.g. total breakdown of a hard disk) are immediately visible, while others (e.g. "silent" block faults on hard disks) are *latent*, i.e. only detectable after active inspection. This implies that low fault detection time (for latent faults) is a very important parameter when designing digital archives.

Data grids as preservation platforms. In [2] the term *data grid* is introduced to describe an integrated architecture common to many systems handling large-scale, geographically distributed datasets. The paper [5] describes data grids as software systems that support the creation of collections that span storage systems located in multiple administration domains. It also discusses how data grids can incorporate commodity-based disk caches, *Grid Bricks*. A Grid Brick is a storage appliance with a particular software stack already installed on it. [5] particularizes the definition to mean a commodity disk cache on which data grid technology is installed to provide a logical name space. The Grid Brick is an extension of the idea of CyberBricks. A CyberBrick is a stand-alone system, containing not only the disk but also the associated controlling CPU and network access. Grid Bricks integrate data grid management services with CyberBrick commodity hardware.

The paper [4] analyses requirements for systems that support the long-term storage of digital records (called *preservation environments*). Data grids based on Grid Bricks are recommended for building preservation environments that minimize the risk of data loss and preserve authenticity and integrity of stored data.

Their opinions are backed by the conclusions of [1]. Unlike more traditional storage management technology, data grids federating storage on large numbers of GridBricks are able to keep several copies of each data object online and thus efficiently inspectable. Constant monitoring and comparison of online copies makes it possible to keep fault detection time low.

Example preservation environments based on data grids. [4] also describes an example implementation of the suggested architecture. Other examples of preservation environements building on the same philosophy include LOCKSS, the Internet Archive and Netarkivet.

LOCKSS is a free, open-source application aimed at preserving access to academic materials that a library licenses from a publisher website ([6]). LOCKSS applications communicate via network to help each other reestablish damaged or lost copies of materials. Much attention is given to the prevention of malicious attacks on this network

[1] This work was presented at *MSST2006*, the 14th NASA Goddard - 23rd IEEE Conference on Mass Storage Systems and Technologies, May 15-18, 2006, College Park, Maryland USA

communication. The LOCKSS strategy against attacks is based on a voting and reputation system and insisting on slow communication (preferring security to speed).

The Internet Archive[2] has been archiving the World Wide Web since 1996. Their data grid relies on Grid Bricks and currently stores more than 500 terabytes of materials. The Internet Archive architecture was a strong inspiration when Netarkivet's own data grid was developed.

The goal of Netarkivet is to collect and preserve Danish cultural heritage on the Internet. We currently have four servers constantly crawling Danish web pages. The crawled pages are stored in a data grid specially developed for the purpose of preserving these materials for the long term. The main requirements for the design of this data grid has been securing the stored data against risks of loss and alteration. The paper [3] describes Netarkivet's data grid (in terms different from the ones used in this paper) and reports on a number of computer simulations that were run in order to predict the longevity of the archived data (the so-called *mean time to failure* of the archive).

1.2. Outline

This extended abstract relates to those procedures in Netarkivet's data grid that allows the system to recover from events that cause damage to or loss of data on a single Grid Brick or server. It describes ongoing work to review, improve and extend the implemented set of recovery operations. Section 2 explains the architecture of Netarkivet's data grid with focus on the preservation aspects. Section 3 provides a formal (mathematical) model of the recovery operations mentioned above. Based on this model, Section 4 goes through a number of scenarios related to data loss/damage and describes if and how each scenario can be handled by the currently implemented recovery operations. The last section concludes on the analysis itself and on the method of reviewing recovery operations by developing a formal model. It also discusses the further work that needs to be done.

2. The Netarkivet data grid

2.1. Architecture

Netarkivet's data grid is organized in three layers. On the lowest layer, it consists of a number of Grid Bricks, each one running the same server application, called the *GridBrickServer*, that manages the data stored on that Grid Brick. In the middle layer, each Grid Brick joins one subgrid. The software application that federates GridBrickServers into a subgrid is called the *SubgridServer*. In the top layer of the architecture, subgrids are joined into the complete data grid of Netarkivet. The central software application that federates subgrids into a single data grid is called the *RepositoryServer*. The RepositoryServer manages a *content index* which is essentially a register that contains the name and MD5 digest (sometimes called checksum or fingerprint) of every file stored in Netarkivet's grid.

(Note: in [3] the GridBrickServer application is named "BitArchiveServer" while a subgrid is called a "BitArchive").

Every subgrid manages a copy of every data file stored in Netarkivets data grid. This propagation is handled by the RepositoryServer's *store()*-operation which we shall not discuss in this paper (although it does build on the techniques described below). None of the subgrids are considered primary.

Subgrids are geographically separate, so that most "external" events (fire, major power outages, natural catastrophies etc.) should only affect one subgrid. Subgrids are also administratively separate, so that the harmful action of one system administrator (whether intentional or by accident) can only affect one subgrid.

The current Netarkivet installation. In the current installation Netarkivet has two subgrids: One in Copenhagen maintained by the Royal Library of Denmark and one in Aarhus maintained by the State and University Library. Netarkivet decided to make the two subgrids different in architecture to minimize preservation risks related to systematic errors in equipment or operating systems. The Copenhagen subgrid is running Windows XP, while the Aarhus subgrid is Linux-based.

2.2. Preventing, discovering and recovering from storage failures

No direct contact with the file systems. One of the design decisions of Netarkivet's data grid was to discourage any direct interactions between system administrators and the file systems on the Grid Bricks. A command line login is a powerful tool that enables its user to accidentally delete large amounts of data in very short time.

A consequence of this decision is that the software (the GridBrickServer) must support all necessary operations on the file system, but with a less failure-prone interface than the command line.

Constant, automated monitoring. Some of the events that can damage stored data are immediately visible, e.g. the complete breakdown of a disk will normally be visible through standard hardware monitoring systems. Other types of events are latent (as described in [1]) and must be discovered through more active monitoring operations.

[2] http://www.archive.org/

One of the great advantages of online data grids as a platform for preservation is that such error discovery operations can be performed automatically and efficiently, unlike preservation systems based on near-line or offline storage.

In Netarkivet, active fault detection is performed through batch jobs supported by the SubgridServer and the GridBrickServers. Batch jobs are executed on both Netarkivet's subgrids on a fixed time schedule.

Recovery operations: manual accept, programmatic execution. As mentioned above, the system implements a number of operations to fix discovered problems without logging on to the affected servers. These operations have been thoroughly tested in a safe environment.

The operations are all 100% automatic but cannot be initiated without a manual accept (clicking "OK" in the user interface). This decision was made to prevent an error in the software to suddenly modify large amounts stored data.

The three operations supported at the moment are: Restoring a file missing in one subgrid, deleting a damaged file from one subgrid, and correcting a wrong entry in the cental content index.

A precise model of these operations is given in the next section.

3. A formal model of Netarkivet's recovery operations

Simplifications. To save space and keep focus on the central points we will make a few simplifications before modelling the Netarkivet system:

- The following model is specific to 2 subgrids (and 1 conent index). Our RepositoryServer is able to handle more subgrids, but describing the operations in full generality would be rather more complex. We name the two subgrids *east* and *west* respectively. We refer to the content index as *idx*.
- Each subgrid may (by error) contain several identical copies of a given file. This is a waste of resources but not in itself a threat to the preservation and we do not model this situation.

3.1. File preservation status

Each operation acts on a single file f in the data grid. Before carrying out any other action, the system checks the current status of the file on both subgrids and in the content index. This ensures that recovery actions are not performed on the basis of information (from a batch job) that may be several days old.

Model. The status information collected in this prephase consists of the following three attributes:

$D_{idx}(f)$ The MD5 digest of f found in the content index (if any was registered).

$D_{east}(f)$ The MD5 digests of all occurrences of f in subgrid *east*.

$D_{west}(f)$ The MD5 digests of all occurrences of f in subgrid *west*.

We model each of these three attributes as a set of bit sequences. The implementation of the content index ensures that $D_{idx}(f)$ contains at most one element.

Status examples. The desired status of a file f is:

$$\begin{aligned} D_{idx}(f) &= \{d\} \\ D_{east}(f) &= \{d\} \\ D_{west}(f) &= \{d\} \end{aligned}$$

for some bit sequence d. This status indicates that both subgrids have excatly on copy of f, and that this copy has the same MD5 digest that was registered in the content index when f was originally stored.

If the file f was lost in subgrid *east* (say, due to a broken disk on the Grid Brick that stored it), the picture would look like this:

$$\begin{aligned} D_{idx}(f) &= \{d\} \\ D_{east}(f) &= \emptyset \\ D_{west}(f) &= \{d\} \end{aligned}$$

3.2. The recovery operations

With two subgrids, the Netarkivet data grid support five distinct recovery operations:

$fix_{idx}(f)$ Corrects a wrong entry for file f in the cental content index.

$del_{east}(f)$ Deletes a damaged file f from the *east* subgrid.

$del_{west}(f)$ Deletes a damaged file f from the *west* subgrid.

$cp_{east}(f)$ When file f is missing in *east*, restores it by copying f from *west*.

$cp_{west}(f)$ When file f is missing in *west*, restores it by copying f from *east*.

Each of these operations requires certain conditions to be satisfied when it is applied. For instance, we do not allow a file f to be restored if the remaining copy has an MD5 digest that does not agree with the one registered in the content index. These conditions and the effects of applying each operation are modelled in Table 1.

Table 1. Recovery operations in Netarkivet. For each operation we list the conditions for applying it and its effect when applied.

Op.	Preconditions	Effect
$fix_{idx}(f)$	$D_{idx}(f) = \{d'\}$ $D_{east}(f) = \{d\}$ $D_{west}(f) = \{d\}$	$D_{idx}(f) = \{d\}$
$del_{east}(f)$	$D_{idx}(f) = \{d\}$ $D_{east}(f) = \{d'\}$ $D_{west}(f) = \{d\} \vee \lvert D_{west}(f)\rvert \neq 1$	$D_{east}(f) = \emptyset$
$del_{west}(f)$	$D_{idx}(f) = \{d\}$ $D_{east}(f) = \{d\} \vee \lvert D_{east}(f)\rvert \neq 1$ $D_{west}(f) = \{d'\}$	$D_{west}(f) = \emptyset$
$cp_{east}(f)$	$D_{idx}(f) = \{d\}$ $D_{east}(f) = \emptyset$ $D_{west}(f) = \{d\}$	$D_{east}(f) = \{d\}$
$cp_{west}(f)$	$D_{idx}(f) = \{d\}$ $D_{east}(f) = \{d\}$ $D_{west}(f) = \emptyset$	$D_{west}(f) = \{d\}$

4. Failure scenarios and recovery

This section will discuss a number of scenarios in which Netarkivet's data grid will yield a status for a file f that is different from the desired one. In each scenario we use the above model to discuss the possibilities of recovering from the faults using the five recovery operations.

Some of the scenarios may seem quite unlikely, but when building a system for long-term preservation one must consider even very improbable event.

It is conjectured that the below list covers all cases that can occur in our model of the system. A formalization and proof of this conjecture is work in progress.

Disk failure on a Grid Brick. A broken disk on a Grid Brick will result in a number of files having the following status (assuming the Grid Brick was part of the *east* subgrid:

$$D_{idx}(f) = \{d\}$$
$$D_{east}(f) = \emptyset$$
$$D_{west}(f) = \{d\}$$

In this scenario, the $cp_{east}(f)$ operation can be applied and it will reestablish the desired status for each file.

Bit rot on a Grid Brick So-called "bit rot", unreadable disk sectors and other similar media failures (see [1]) will cause some file f to have a status like the following:

$$D_{idx}(f) = \{d\}$$
$$D_{east}(f) = \{d'\}$$
$$D_{west}(f) = \{d\}$$

In this scenario, Netarkivet can recover by applying the $del_{east}(f)$ operation followed by $cp_{east}(f)$.

Bit rot in the content index. Bit rot and similar media failures affecting the content index will cause some file f to have a status like the following:

$$D_{idx}(f) = \{d'\}$$
$$D_{east}(f) = \{d\}$$
$$D_{west}(f) = \{d\}$$

In this scenario, Netarkivet can recover by applying the $fix_{idx}(f)$ operation.

Irrecoverable scenarios. Some failure scenarios – in particular "double faults" where both copies of a file are damaged or lost – are irrecoverable by nature. As a matter of principle, Netarkivet considers the situation where no MD5 digest for a file f is reported by more than one system entity

$$D_{idx}(f) \cap D_{east}(f) = D_{idx}(f) \cap D_{west}(f) = D_{east}(f) \cap D_{west}(f) = \emptyset$$

as irrecoverable. The rationale for this principle is that it requires human judgement to determine which (if any) copy of f is undamaged.

Deletion of the content index. This will result in many files having $D_{idx}(f) = \emptyset$. None of the currently implemented operations can be applied in this situation.

Appearance of one or more extra copies of a file in a subgrid. If the copies are not identical, this should result in a file f with e.g. $\lvert D_{east}(f)\rvert > 1$. None of the currently implemented operations enables the removal of a "bad" file, if the given subgrid has more than one copy of the file.

A special case with multiple copies in a subgrid. In the particular case where the content index has registered a wrong MD5 digest for a file f and one subgrid (say, *east*) contains an undamaged copy of f *and* one or more damaged copies of f,

$$D_{idx}(f) = \{d'\}$$
$$D_{east}(f) = \{d, d'\}$$
$$D_{west}(f) = \{d\}$$

the operation $del_{west}(f)$ can actually remove an undamaged copy of f from the subgrid *west*! This bug in the design of

cp was not revealed until the analysis presented here was worked out. The bug is expected to be fixed in the next release of Netarkivet's software.

5. Conclusion and next steps

Status and developments in the software of Netarkivet The conclusion of the analysis is that the supported operarions enables the system to recover from several events that cause data loss or damage. The analysis has also pointed out the scenarios that Netarkivet's system should be extended to handle, and it has uncovered an actual bug in the current version of the software running on the data grid.

The author is currently working on establishing that the list of scenarios discussed in Section 4 is complete, i.e. that every possible files status has been considered. When a proof of completeness has been worked out, the plan is to discard with the simplifications listed in Section 3, i.e. to generalize the analysis to cover

- Installations with more than two subgrids and more than one content index.
- Subgrids containing several identical copies of a given file f.

The next step will be to design and implement an improved and extended set of recovery operations that cover all scenarios in a satisfiable manner.

Recovery in other data grids As discussed in the introduction, data grid technology is becoming popular as the basis of systems intended for long-term preservation of digital materials. While the operations described are proprietary to Netarkivet's systems, most preservation environments based on grid technology should implement a set of similar operations for recovering after loss of or damage to a copy of a file. It should be possible to adapt the above analysis to other such systems. Even if an adaption of the analysis itself turns out to be impractical, the author would recommend preservation environment designers to make their own simple but precise, formal analyses of data loss scenarios. The effort has been very worthwhile in the case of Netarkivet. A preservation environemt's ability to recover from failures is essential to its purpose, and a precisely formulated design can prevent many bugs and misunderstandings in the implementation and use of that functionality.

References

[1] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. J. Giuli, and P. Bungale. A fresh look at the reliability of long-term digital storage. http://arxiv.org/abs/cs/0508130, Aug. 30 2005.

[2] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, (23):187–200, 2000.

[3] N. H. Christensen. Preserving the bits of the Danish internet. In J. Masanes and A. Rauber, editors, *Proceedings of the 5th International Workshop on Internet Archiving (IWAW05)*. http://www.iwaw.net/05/papers/iwaw05-christensen.pdf, 2005.

[4] R. W. Moore, J. JáJá, and R. Chadduck. Mitigating risk of data loss in preservation environments. In *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005)*, pages 39–48, 2005.

[5] A. Rajasekar, M. Wan, R. Moore, G. Kremenek, and T. Guptil. Data grids, collections, and grid bricks. In *IEEE Symposium on Mass Storage Systems*, pages 2–9, 2003.

[6] D. R. V Reich. LOCKSS: A permanent web publishing and access system. *D-Lib Magazine*, 7(6), June 2001.