Preserving the bits of the Danish Internet

NIELS H. CHRISTENSEN

Dept. of Documentation and Digitalisation, Royal Library of Denmark
Postbox 2149, DK-1016 Copenhagen K, Denmark

nhc@kb.dk
Phone: (+45) 33 47 46 90
Fax: (+45) 33 93 22 18

One of the many challenges in large-scale web archiving is long-time preservation of the amounts of data generated by the harvester. We describe the bit preservation solution chosen by Netarkivet. We then define a programmatic, probabilistic model of hardware failures and repair operations in the solution. The mean time to failure of this model is then computed in a number of experiments based on simulation.

Keywords: web archiving, bit preservation, mean time to failure, simulation of probabilistic models

1. Introduction

In December 2004, the Danish parliament passed a new legal deposit law, defining new rules and responsibilities from July 2005. The new law calls for large-scale harvesting of the Danish part of the Internet for the purpose of preserving our cultural heritage. The operations associated with the new law are executed by The Royal Library and The State and University Library in Denmark. The two libraries work closely together on the establishment of the whole archive called 'Netarkivet' and develop, in collaboration, strategies and software for the collection, archiving, preservation and access of material.

In this paper we focus on the preservation of the materials, in particular the *bit preservation* of the harvested files. In other words, we shall describe and analyse the software and hardware systems that are responsible for storing ingested files in such a way that they all can be retrieved bit by bit for decades or centuries. We call this subsystem our archival, digital repository (or most often just "the repository").

Our aim here is to describe the design of Netarkivet's repository and to illustrate how we modelled it and performed a statistical evaluation of its robustness over time; how long it will take before some disaster causes it to lose archived data. We believe that our approach and experimental setup (with modifications) could be useful for other repository designers. We also discuss the results of the experiments we did with our statistical evaluation in order to examine different means of improving our system.

The reader should note that while the work in this paper was done as part of the establishing a repository for Danish Internet materials, there is nothing inherently Danish in the designs or results obtained.

1.1 Related work

Every designer of an archival, digital repository should find [Dpc05] useful. This technology watch report by the DPC describes briefly but clearly how the British Library (BL) approached the challenge of setting up their long-term, digital repository. The report discusses the overall issues considered and the requirements that were formulated. It also summarizes the interaction between BL and the potential suppliers, the design choices made and their rationales.

The inspiration for the modelling and simulation work presented in this paper is the report [Crespo00ext] by Crespo et al. (but see also the briefer paper [Crespo00] or the Ph.D. thesis [Crespo03]). Because this work is central to understanding our own approach, we end this section summing up the report in some detail.

The report briefly reviews the main sources of data loss in archival repositories: media decay and failure, component (e.g. media or format) obsolescence, human and software errors, and external events (fires, earthquakes etc.) and discusses the main strategies for avoiding failures. The authors then move on to explain their model of archival repositories. The model is focussed on the technical systems (unlike, but compatible with, OAIS [Oais], which also models organization, not to mention many other aspects of archives than preservation). An archival repository is seen as having two parts: a *data store*, which is not in itself fault-tolerant, and an *archival system* that handles ingest and access to documents in the data

store and adds fault-tolerance to the data store by performing *failure detection*, *damage repair*, and *failure prevention*. Each document in the archival repository has a number of *manifestations*; ways in which it can be rendered to the user. For example, manifestations of a given document may vary in the device on which it is rendered (screen/printer), the logical format it is delivered in (pdf/postscript) and at which physical storage site it is read. A document is considered as lost, when it has no more manifestations, i.e. when there is no known ways in which it can be rendered to a user. The report introduces a model in which a document can be in one of 7 states: created (but not yet archived), archived (but not necessarily accessible), accessible, damaged, damage detected, lost, retired. These are the main states - a given repository model can use fewer or more states as applicable. For instance, in the repository of Netarkivet, all archived documents are accessible, and no documents are ever retired.

When modelling an archival repository, Crespo et al. defines the exact states that a document can be in and, for each (relevant) transition from one state to another, the probability of this transition happening for a document on a given day. The authors of [Crespo00ext] built a tool, ArchSim, which was able to simulate such archival repository models. From a number of simulations, the system could estimate the *mean time to failure* (MTTF) of the modelled archival repository, i.e. the average time it takes before an archive repository fitting the input model experiences its first (irrecoverable) data loss. The paper describes some details of ArchSim and finally demonstrates the whole method by analyzing a realistic case study.

1.2 Overview of this paper

In Section 2, we describe the design of the digital repository used by Netarkivet. Section 3 goes on to designing a probabilistic model of the repository in the style of [Crespo00ext]. It also touches on the tool that we wrote to evaluate the MTTF of our model. Section 4 discusses the results of the simulations that we did with our evaluation tool. Section 5 concludes and points to future work.

2. The digital repository of Netarkivet

In this section we describe the digital repository that stores the material of Netarkivet. The data that is ingested into our repository is generated by harvesting the Danish web. The repository contains only one type of files: ARC files ([Arc]) as written by the Heritrix web harvester software and its libraries ([Heritrix04]). Heritrix is expected to implement and use the upcoming WARC format being defined by the International Internet Preservation Consortium ([Iipc]). When this happens, our repository will store WARC files, but to simplify our presentation, we call all our files ARC files. In Netarkivet, most ARC files will be close to 500MB in size. An ARC file contains a sequence of harvested data objects, each one preceded by a small ASCII header.

In our repository each stored ARC file is identified by its file name, e.g. "foo.arc". The file name is not a path, i.e. it does not contain "/" or "\". We guarantee that only one file is ever stored under a given file name (although we will have several copies of the file under that name).

2.1 Overall architecture

The repository of Netarkivet is located at two physical sites: one at The State and University Library (SB) in Århus and one at The Royal Library (KB) in Copenhagen, see Figure 1. Every file in the repository is stored at both sites - indeed a file is not considered as stored in the repository before two uploads have been confirmed. The systems that store ARC files at the two sites are called BitArchives (we use this specific term to avoid confusion with other definitions of "bit archive"). In the near future, each BitArchive will have a capacity of 20TB.

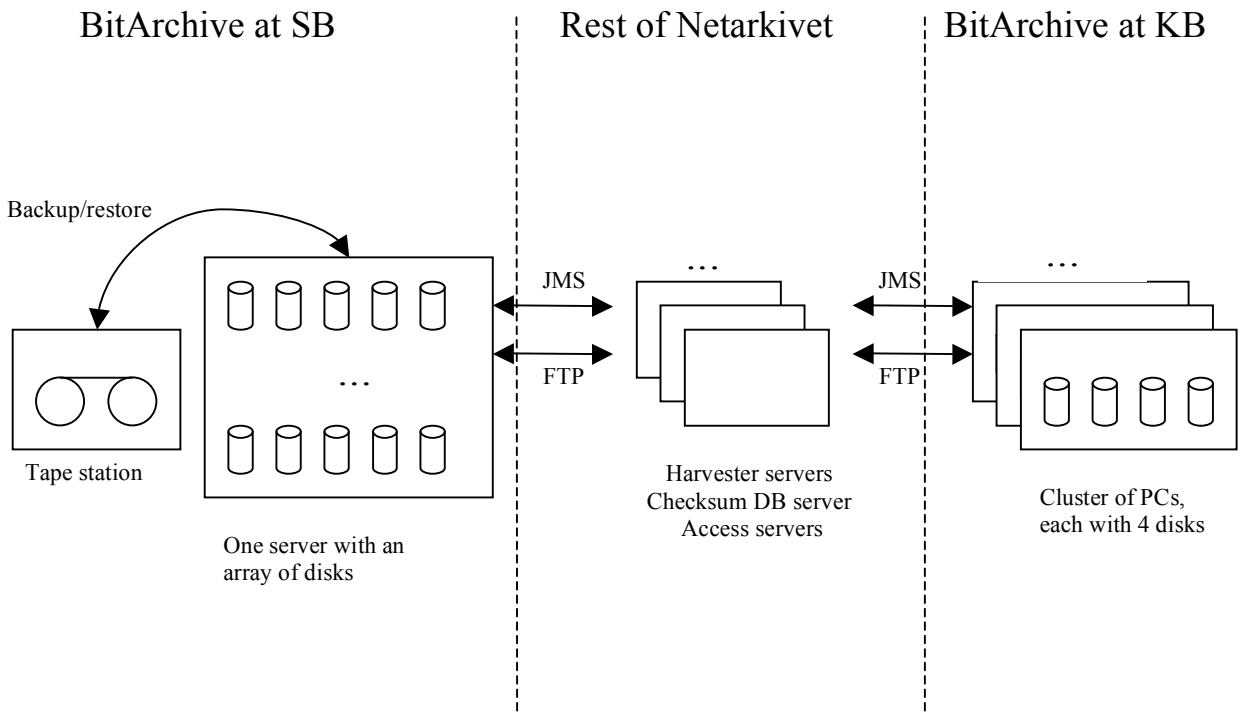

Figure 1: The repository contains two 'BitArchives', one at SB and one at KB.

The BitArchive at KB is optimized for fast data access. It is inspired by the solution used by Internet Archive. The data is stored on a cluster of inexpensive PCs, each one equipped with 4 independent, commodity-type disks. As more data is stored, more machines will be added to the cluster. As disk capacity increases, the disks in new machines will be larger than the ones in the current setup.

The BitArchive at SB is optimized for robustness. The data is stored on one large array of IDE ("Intelligent/Integrated Drive Electronic") disks running RAID ("Redundant Array of Inexpensive Disks"). The exact RAID configuration is still to be determined at the time of writing, but in our model we shall assume that groups of 5 disks can survive one disk crash without data loss. Every file written to the disks is also backed up on tape. The tapes are stored offline in a safe. As more data is stored, more disks will be added to the array, and more tapes will be used for backup. As disk capacity increases, the added disks will be larger than the ones in the current setup. The same applies for tapes.

The stability of each server in the BitArchives is of great importance to Netarkivet. The software installed, ports opened etc. is therefore minimal on these machines. They are placed on guarded subnets of each institution's network.

The Bitarchives are coordinated through a central repository server that also contains a database of MD5 checksums ([Md5]) of all ingested files.

2.2 Software

All software systems are implemented in Java. Each machine in each BitArchive runs exactly one Java Virtual Machine (JVM). This JVM executes an application called the BitArchiveServer, and controls all reading and writing in the data areas of the computer it is running on. The BitArchiveServers communicate with the rest of the system through JMS (Java Message Service, a part of the J2EE framework) and FTP (File Transfer Protocol). JMS is used for robustly sending "atomic" requests and responses, while FTP is used for transporting larger amounts of data (like ARC files). We are currently planning on replacing FTP with the secure shell protocol (SSH), but the JMS component will still be used for reliable messaging between applications.

As an example, a BitArchiveServer may receive a JMS message requesting permission to upload a given (newly harvested) ARC file. The message itself does not contain the ARC file itself. Rather, it contains information like a file name and the IP- and port numbers of an FTP server. Upon accepting this request, the BitArchiveServer will contact the specified FTP server and download the specified file. The downloaded file is written to a random disk among those with free space left. When this operation has been completed, the BitarchiveServer sends back a confirmation via JMS. The upload operation will be followed by more messages crosschecking the checksum of the uploaded file with the central database before the ingest operation is considered complete.

2.3 Hardware failures: discovering and recovering

All of the hardware used by Netarkivet supports online, automatic surveillance so that the operators get immediate notification if a disk breaks down. Media decay (sometimes called "bit rot") resulting in single bit losses may not be discovered by these means, so we perform regular scans of all online files and verify their checksums against the one recorded in our central database. While it is possible to verify the readability and contents of a tape and correct lost content, we do not have documented procedures for this at the moment.

If the KB BitArchive loses its copy of a file, we may recover by

1. Checking that the copy at the SB BitArchive is OK, by crosscheking with the checksum database.
2. Copying the file from the SB BitArchive to the KB BitArchive.

Note that in the case of a disk crash, this operation will be performed for every file that was saved on the crashed disk. The recovery operation is performed by software, rather than e.g. manual operations on a command line, to minimize the risk of typing errors and such, but the operation will only be started after a manual confirmation. This is to minimize the risk of software errors causing erroneous corrections of a large number of files in the archive.

If the SB disk copy is damaged, it may be recovered from the tape copy, providing the tape copy is OK. If it is not, we may recover from the KB copy using the algorithm described above.

If the checksum of a file is lost in the database, if the data is corrupted or the whole database should be lost, we may recover by

1. Retrieving the MD5 checksum of the existing copies in the two BitArchives and checking that they agree.
2. Writing this checksum into the database.

If both BitArchives lose their copies of the same file, it is irrecoverably lost. The file is also considered lost if one BitArchive loses its copy and the copy at the other BitArchive does not have a checksum that agrees with the one stored in the central database. In this case, the apparently corrupted remaining copy will be removed from the archive but stored elsewhere. The responsible archivists can then review its contents and decide whether it should be reingested to the archive. Nonetheless, we count this situation as data loss.

2.4 Other failures: discovering and recovering

In this section we briefly consider some of the failure types that are not in the main scope of this paper.

All repositories, even the non-digital ones, are prone to catastrophies like fires and earthquakes. The physical sites of KB and SB seem quite well-protected against such incidents, but our main protection is in the geographical redundancy of the repository. Only a disaster devastating both Århus and Copenhagen (some 200 kilometers apart) could wipe out the repository completely.

Another threat is errors and weaknesses inherent in a technical architecture, like the Y2K (year 2000) bug. To counter this threat, our two BitArchives use different hardware types from different vendors, and will run different operating systems (Windows and Linux) and different implementations of Java.

System operators are not infallible either. We therefore ensure that no one person has rights that make her able to cause irrecoverable data loss in the repository. We also restrict physical access to the BitArchive servers.

Probably one of the largest risks is software bugs. There is no final recipe to avoid this threat, but we have employed a number of quality assurance techniques to minimize the risk of critical bugs in our software. We also test every software update in a completely separate testing environment before it is deployed into our production system.

3. A simulation model of the repository

In this section we describe our *standard model* of Netarkivet's repository. The purpose of the model is to describe how files in the repository may be damaged -lost, but not irrecoverably so - and repaired, that is recovered from damage. The model is probabilistic and will allow us to run a simulation of the repository from its launch (day 1) until it incurs its first irrecoverable data loss. By running a number of simulations and computing the average number of days passed from repository launch until the first irrecoverable data loss we get an estimate of the MTTF of the repository. In our experiments, in Section 4, we shall compute such an estimate not only for our standard model, but also for a number of variations thereof.

3.1 Principles

The sources of failures in a digital repository are legion and most of them are very hard to estimate probabilities for. To keep our model simple enough to understand - and simulate efficiently - we shall focus only on a few ones and leave the rest out of our model. The consequence of this approach is of course that our estimated MTTF will be too optimistic, because removing a failure source from the repository will always lower the risk of failures on any given day of the repository's operations. It should be noted, though, that the real repository at Netarkivet actively addresses several of the failure sources that we do not model, as explained in Section 2.4. Leaving out the failure source also means leaving out its remedy, so the estimates are not necessarily altogether unrealistic. But even if the actual numbers are marked with considerable uncertainties, our simulations will still allow us to study certain characteristics of the features that we *do* model, because the uncertainties apply equally to all variations of our model.

On the failure source side, our primary focus will be media breakdowns, i.e. the effect on the repository when one or more disks and/or tapes simply stop being readable due to mechanical or similar problems with the medium in question. On the repair side we will model the strategies that Netarkivet employs to address the problem of media breakdowns, i.e. having several copies on disks and tape, having a controlled way of recovering lost copies from the other copies, and using RAID technology on one of the installations.

Figure 2 gives an overview of the assumptions that lie behind our standard model of Netarkivet's repository. Each assumption reflects some feature of our repository that we have chosen not to model; something that is not realistic, but which simplifies our simulation.

1. No fires, earthquakes, etc.
2. No inherent platform errors like Y2K
3. No media obsolescence
4. No loss of data before or during ingest
5. No media decay - only total breakdowns
6. Busy weekends - disk failures are detected and acted upon within same day
7. No network failures
8. No software errors
9. No operator errors
10. No format obsolescence
11. Every day, each disk has a failure probability of 1:1096. [Most hardware has a support period of around 1096 days.]
12. Every day, each tape has a failure probability of 1:1096.
13. Repair operations may transfer a max. of 600GB per day
14. When a disk fails, it is always full
15. Static model: no ingest is made after day 1

Figure 2 : An overview of the assumptions behind our standard model of the Netarkivet repository.

The first five of the assumptions concern factors that we explicitly addressed when designing our repository, but which we deliberately chose not to study as part of our experiments. The first three - natural disasters, Y2K-like problems and media obsolescence - seem very hard to make any kind of probabilistic model for. In our repository, we address these issues by geographical redundancy, by using different platforms and architectures in the two BitArchives, and by a requirement that all disks and tapes must be migrated within their period of supplier support. The following two assumptions - concerning data loss before ingest and media decay - we feel able to model, but we decided to leave studies of these factors for future work.

Assumptions no. 6 and 7 are real challenges that we only partly address. We do not have the means to ensure 24-hour/7-days surveillance of our system, so there will be times when damaged data will take longer time to repair than others. Large-scale network breakdowns make us unable to repair damaged data by transferring between BitArchives. If such problems were to last for longer periods of time - which is possible but not likely - we might have to close down systems in order to minimize the risk of further failures.

Assumptions 8-10 are the ones that we would most like to address in future work with our model and simulations, as these factors are often quoted as the biggest sources of data loss. There are methods for measuring and minimizing the risks of errors in software and human errors, but we know of no study that systematically applies these methods to archival, digital repositories. Such a study would be extremely useful, but it would also be much more advanced than the experiments presented in this paper. Format obsolescence and (more generally) the problem of logical preservation deserves a study of its own.

Turning to assumptions 11-12, they concern the lifetime of disks and tapes respectively. Media lifetimes are the topic of many reports, and the original ArchSim had some advanced probability distributions for predicting media breakdown. We choose, for simplicity, a naïve but certainly pessimistic assumption: each day, for each disk or tape in the system, a 1096-sided die is thrown. If the result is a 1, the medium in question breaks down. The value of 1096 is arguably too pessimistic; our assumptions give any disk or tape an expected lifetime of 3 years. We expect a longer durability, especially of the tapes. Assumption no. 13 asserts that the network bandwidth is limited. The estimate of 600GB/day is based on practical experiments on our network. Assumption 14 - that disks are always full when they break down - is really just Murphy's Law in action.

The last assumption states that we only ingest data once, on the day when the repository is launched, and no data is added to the repository afterwards. This may seem to contradict our intention to model a repository for a web archive that performs large-scale harvesting of the national web. Such a repository should expect a continuous stream of ingested data, and indeed we expect an exponentially *increasing* amount of data to be harvested each year.

The point of our last assumption is that we also expect an exponential increase in the size of disks and network bandwidth. Given these expectations, our last assumption really boils down to the following:

- The amount of harvested data, the size of disks and the network bandwidth available to Netarkivet all grow with the same exponential factor.
- Netarkivet will constantly keep up with the improved technology.

Note that the effect of constantly growing disk sizes, from the point of view of repository longevity, is that each year more data can be stored with the same failure rate for the same price as before. In our model, for example, if some amount of data is migrated from 300GB disks to 600GB disks, the expected amount of data lost per day will drop by a factor of two.

In our experiments in Section 4, we investigate whether a static archive (with no data ingested after day 1) and an escalating archive (where the above-mentioned numbers grow exponentially) give similar results in simulations.

In the following sections we show how we built a model of Netarkivet's repository based on all of these assumptions. Section 3.2 explains the possible states of any given file in the modelled repository. Section 3.3 then describes how files go from one state to another during a simulation of the model.

3.2 States in the model

The overall state space is shown in figure 3.

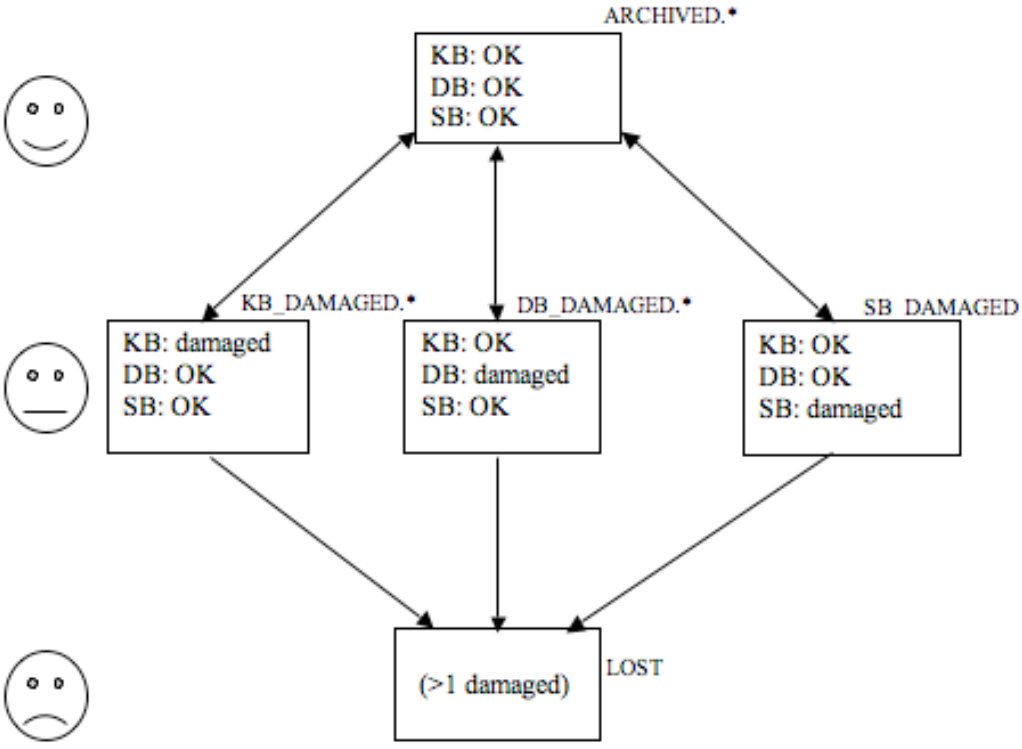

Figure 3:The overall states of a file in our model of Netarkivet. Some states have substates (see the main text); these are suffixed by the wildcard '*'.

Immediately after ingest, the file is in the state "ARCHIVED". In this state, the file is archived in two BitArchives ("KB" and "SB"), and its MD5 checksum is stored in the database ("DB"). Each of these three consituents may lose their copy/checksum of a given file. We say that the local copy/checksum is *damaged*. The corresponding states are "KB_DAMAGED", "DB_DAMAGED", and "SB_DAMAGED". If two of the three constituents lose their copy/checksum of a given file, the file ends up in the LOST state.

As mentioned above, the BitArchive at SB keeps two copies of each stored file, a disk copy and a tape copy. We only consider the SB site as having lost a file if both copies are damaged. This is the reason that some of the states in Figure 3 have substates. The internal state at SB is illustrated in Figure 4.

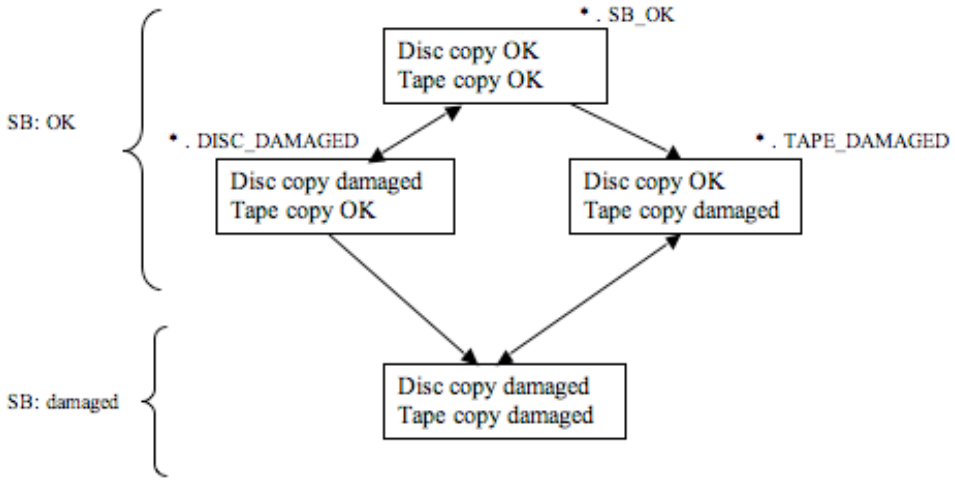

Figure 4:'SB: OK' covers three different situations, because the SB BitArchive keeps copies on both disk and tape.

The reader may notice in Figure 4 that there are no arrows representing the recovery of a file broken on tape. As we mentioned above, we do not have documented procedures for this at the moment, so our model of Netarkivet does not include neither verifying tape content nor recovering lost tape content.

By combining Figures 3 and 4, we get a total of 11 states:

- `ARCHIVED.SB_OK`
- `ARCHIVED.DISC_DAMAGED`
- `ARCHIVED.TAPE_DAMAGED`
- `KB_DAMAGED.SB_OK`
- `KB_DAMAGED.DISC_DAMAGED`
- `KB_DAMAGED.TAPE_DAMAGED`
- `DB_DAMAGED.SB_OK`
- `DB_DAMAGED.DISC_DAMAGED`
- `DB_DAMAGED.TAPE_DAMAGED`
- `SB_DAMAGED`
- `LOST`

3.3 Transitions between states

Our entire model is summed up by the pseudocode in Figure 5. We comment on that code in the following.

```
INGEST: if dayNo == 1
          then add 20000 GB in state ARCHIVED

REPAIR: move all data from state *.DISC_DAMAGED to state *.SB_OK

        repairLimit = 600
        forRepairAtSB = count GB in state SB_DAMAGED
        repairedAtSB = minimum(forRepairAtSB,repairLimit)
        move repairedAtSB GB from state SB_DAMAGED
                                to state ARCHIVED.TAPE_DAMAGED
        remainingRepairLimit = repairLimit - repairedAtSB

        forRepairAtKB = count GB in state KB_DAMAGED.*
        repairedAtKB = minimum(forRepairAtKB,remainingRepairLimit)
        move repairedAtKB GB from state KB_DAMAGED.*
                                to state ARCHIVED.*

        move all data in state DB_DAMAGED.* to state ARCHIVED.*

BREAK:  GBAtKB = count GB in states ARCHIVED.*,DB_DAMAGED.*,SB_DAMAGED
        discsAtKB = (GBAtKB / 300) + 1
        brokenKB = discFailures(discsAtKB)
        move (brokenKB * 300) GB either from state ARCHIVED.*
                                      to state KB_DAMAGED.*
                                   or from states DB_DAMAGED.*,SB_DAMAGED
                                      to state LOST

        GBAtSB = count GB in states *.SB_OK,*.TAPE_DAMAGED
        discsAtSB = GBAtSB / 300
        raidUnitsAtSB = (discsAtSB / 5) + 1
        raidsBrokenAtSB = raidFailures(raidUnitsAtSB,5)
        move (raidsBrokenAtSB * 1500) GB either from state *.SB_OK
                                             to state *.DISC_DAMAGED
                                          or from state ARCHIVED.TAPE_DAMAGED
                                             to state SB_DAMAGED
                                          or from states
                                                      KB_DAMAGED.TAPE_DAMAGED
                                                      DB_DAMAGED.TAPE_DAMAGED
                                             to state LOST

        if raidFailures(1,4) == 1  and  tapeFailures(1) == 1
          then move all data from state ARCHIVED.* to state DB_DAMAGED.*
               move all data from states KB_DAMAGED.*,SB_DAMAGED to state LOST

        tapedGBAtSB =  count GB in states *.SB_OK,*._DISC_DAMAGED
        tapesAtSB = (tapedGBAtSB / 300) + 1
        brokenTapes = tapeFailures(tapesAtSB)
        move (brokenTapes * 300) GB either from state *.SB_OK
                                          to state *.SB_TAPE_BROKEN
                                       or from states KB_DAMAGED.DISC_DAMAGED
                                                     DB_DAMAGED.DISC_DAMAGED
                                          to state LOST

END?:   lostGB = count GB in state LOST
        if lostGB > 0
          then end simulation and output dayNo
```

Figure 5: Pseudocode describing all events occurring in one day's operation of our standard model.

INGEST: As discussed, our basic model is static, in the sense that no files are ingested into the archive after the first day. The exact amount of data in the archive is a parameter that we vary in all our experiments, but in our basic model the value is 20TB, which is the approximate amount of data that Netarkivet expects to ingest during its first year.

REPAIR: There are four operations that repair damaged files in our repository model. First, if the SB disk copy of a file has been damaged, but the copy on tape is intact, it is instantly repaired. Second, if both SB copies of a file are damaged, it is restored from the KB copy (if that is intact). This is done for up to 600 GB per day. Third, if the KB copy of a file has been damaged while SB has an intact copy, the KB copy is restored from the SB copy. This is done for up to 600 GB *minus* the amount of data repaired in the second step. The reason for preferring recovering the SB BitArchive is simply that a file stored only on

KB is more at risk than one stored only at SB and thus more important to replicate. Finally, if the central database has lost the checksum of a file, the checksum is instantly restored.

BREAK: There are four kinds of events that cause files to become damaged or lost. First we estimate the number of active disks at KB as the amount of GB stored in the KB BitArchive divided by 300 (each disk holds 300 GB). The function discFailures() tells how many of these break on the given day by simulating a die-throw per disk (see Section 3.1). For each broken disk, 300 GB of data is chosen randomly among those files that KB has an intact copy of. The state of this data is changed to indicate the new damage. If some of the data was already damaged at SB or in the database, it is now lost, and the simulation will stop when the "END?" stage is reached.

The scenario for disk failures at SB is similar. We let each 5 disks form a RAID unit that can survive one disk crash without data loss. When a RAID unit *does* fail, i.e. when two disks in the same unit break on the same day, the content of all 5 disks is lost. The function raidFailures() simulates that behaviour, given the number of RAID units and the number of disks in each.

The third type of failure event concerns the central database. This database is stored on a 4-disk RAID unit and an ordinary tape backup. If both of these fail, all checksums are lost.

The fourth type of failure event concerns tape failures and is very similar to the KB disk failure scenario. Each tape stores 300 GB. The tapeFailures() function simulates a die thrown for each tape (see Section 3.1).

END?: A simulation ends when at least one file is in state LOST. If no files are in state LOST, the day number is incremented by 1 and the simulation goes through the above steps again.

3.4 The simulation tool

Unfortunately the original ArchSim system is no longer available, so we had to write a simulation system ourselves. The simulation system was written in Java by the author. It builds heavily on ideas from ArchSim, but is in no way as advanced as ArchSim was.

To simulate a repository model, one must define it by writing code somewhat like the one shown in Figure 5, but in Java. The system supplies functionality corresponding to the keywords **move** and **count**, and the functions **discFailures()**, **tapeFailures()**, and **raidFailures()**.

When started, the system runs a specified number of simulations and writes the observed time to failure of each simulation to a file. The system also copies all of its source files in order to document the exact setup leading to the simulations results.

4. Experimental results

In this section we report the results of our experiments with the simulator described above. All experiments were run on an HP ML-370 G3 server with 2 GB RAM and two Intel Xeon 3 GHz processors. One simulation of a repository model results in one time to failure for that repository. Each MTTF given below is the mean based on 250 simulations. The running time of an experiment depends on the MTTF of the given model: Lower MTTF gives lower running time. 250 simulations of our standard model took 1-2 hours to complete. We have not identified the main bottlenecks of the simulator, but we expect that several optimisations could easily be made to the code.

4.1 MTTF of the Netarkivet repository

The first number of interest is of course the basic one: Given the assumptions listed earlier, what is the expected MTTF of our repository? The answer is 144 years. We computed the MTTF for different amounts of data stored in the repository, see Figure 6. The MTTF falls rapidly as the amount of data increases. When the amount is doubled to 40TB, the MTTF drops to 40 years.

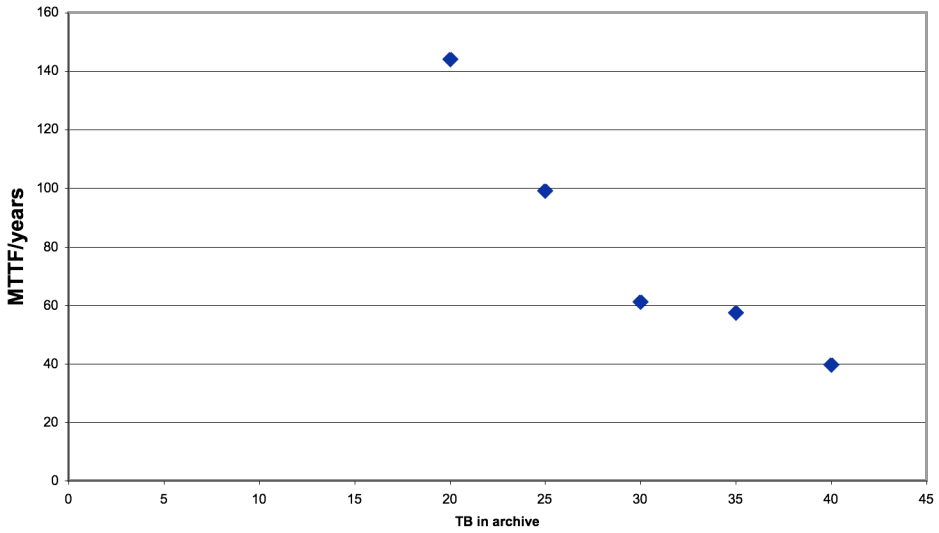

Figure 6:The mean time to failure of our standard repository model for different amounts of ingested data.

4.2 Is the static model reasonable?

We also investigated how the static model compares to one that more directly models continuing harvests and thus continuing ingest. In this alternative model, a number of events take place once per year:

- An amount of data is ingested; this amount is double the amount that was ingested the year before.
- The size of each disk/tape in the system doubles.
- The bandwidth between the two physical sites doubles.

The first year of the archive an amount of 20TB is ingested, and the file size is 500MB. Each disk holds 300GB, and we allow 600GB of data to be repaired over network each day.

The MTTF of this model was also computed. It is 39 years and thus very close to the MTTF of a 40TB repository in our standard model. This result is exactly what we should expect. After a few years, the "escalating" repository will almost exactly be doubling its size each year. For example, in the first four years a total of 20+40+80+160=300TB is ingested, while in the fifth year, 320TB is ingested. In other words, the total amount of ingested data is twice the yearly ingest. If this was set to 20TB from the beginning, the escalating repository should behave as a static model containing 40TB.

4.3 The sum is greater than the whole

It is the hope that the combination of two BitArchives into one repository should have a higher MTTF than each of the constituent BitArchives. Our algorithm for repairing damages is intended to rescue the repository from data losses for a long time. We investigated whether this turned out to be the case. In one experiment with our simulator, we created two alternative models. The first model describes a repository that only includes the KB BitArchive. The second model describes a repository that only includes the SB BitArchive. None of the models include a checksum database, as this would not help the single BitArchive in recovering lost data. Apart from this, all assumptions and parameters are as in our standard model.

The result of computing the MTTF of this two repository models can be seen in Figures 7 and 8.

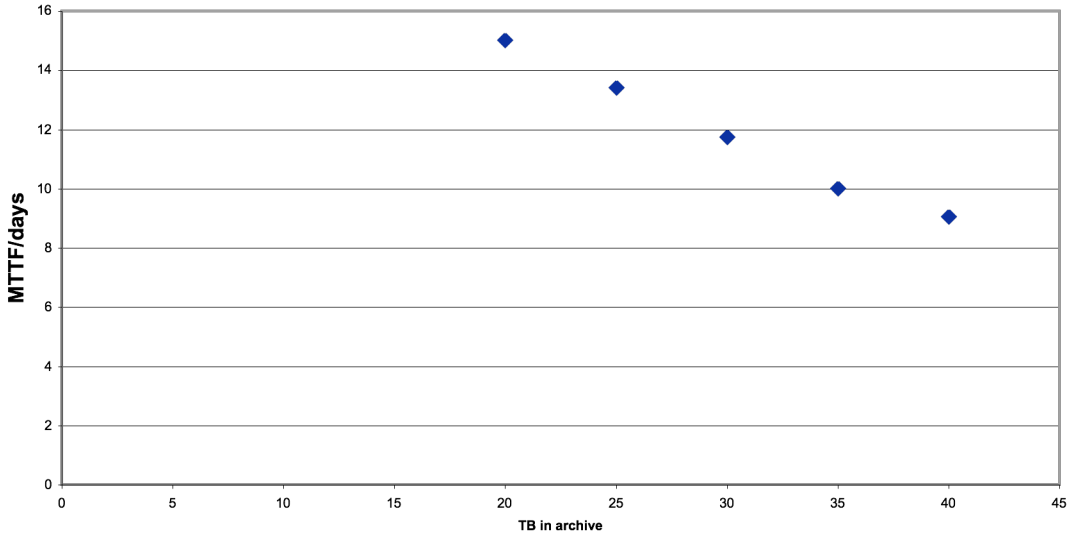

Figure 7:The mean time to failure of the KB Bitarchive by itself. Note that the mean time to failure is measured in days rather than years.

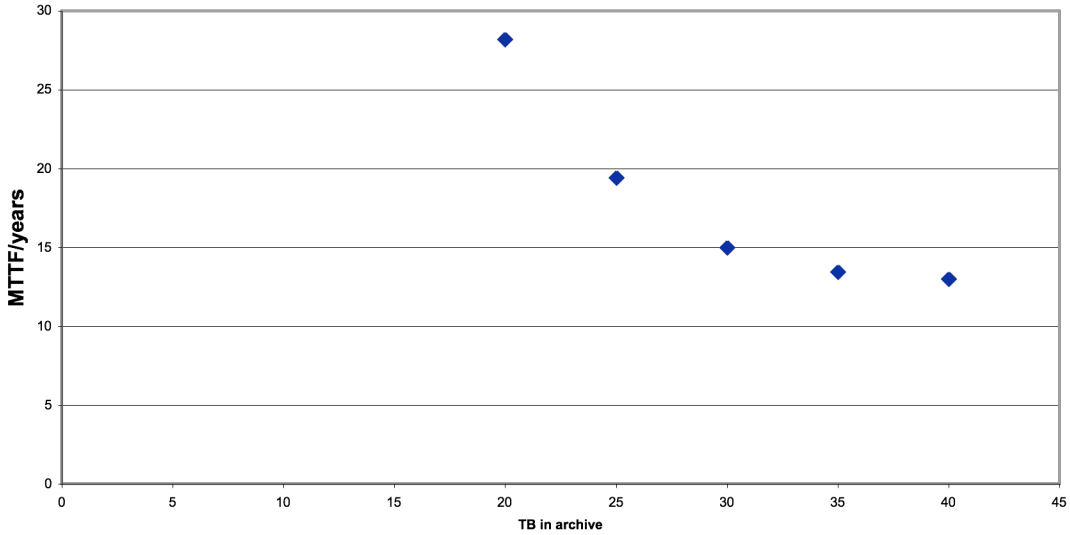

Figure 8: The mean time to failure of the SB Bitarchive by itself.

As one might have expected, the MTTF of the KB Bitarchive is very low. When storing 20TB on our PC cluster, the first data loss should be expected after mere 15 days. This number is almost halved when the amount of data is doubled.

The SB BitArchive - which was designed for robustness - fares much better. It archives 20TB for 28 years without data loss on average. 40TB can be stored for 13 years before the first data loss. These numbers are orders of magnitude larger than the KB numbers, but also significantly smaller than the MTTF of the combined Netarkivet repository. Thus we have been confirmed in our belief that the sum is greater than the whole; two BitArchives put together using our scheme with the checksum database are much more robust than a single BitArchive.

4.4 The impact of RAID

One key point of our design discussions in Netarkivet has been whether to invest in RAID on the PCs at KB. The price of each PC would be higher, but the failure rate of the KB BitArchive would be expected to drop significantly. We decided to investigate the impact of investing in RAID. In an alternative model, based on a concrete suggestion by our hardware responsible, each PC has 6 disks of 250GB each. Two disks must fail on the same day to damage the copy at KB, but this damages the contents of all 6 disks. We count each PC as 1,5 TB of storage even though there should be a reduction of about 10% because of the RAID overhead. When applying these changes to the model of the KB BitArchive presented above, we get a very different MTTF than before, see Figure 9.

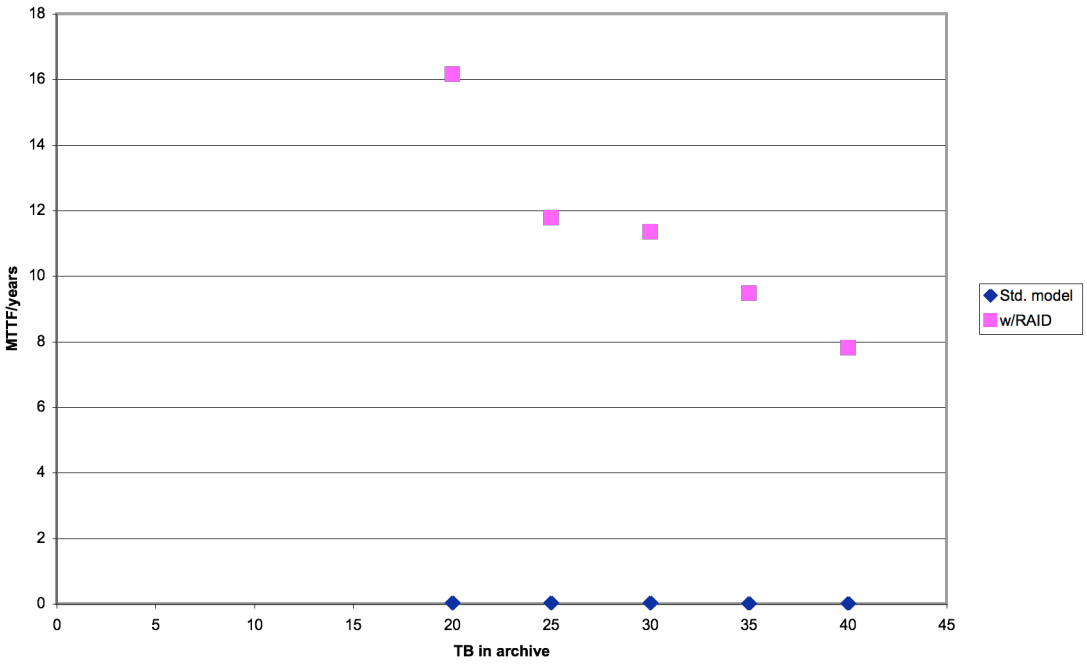

Figure 9: The mean time to failure of the KB Bitarchive by itself, with and without using RAID on the individual PC.

For 20TB of stored data, the KB BitArchive now has a MTTF of 16 years. Compare this to the original KB standalone model with a MTTF of 15 days. These numbers have a direct consequence: much fewer repair operations involving network transfer from the other BitArchive will be needed when RAID is used on the PCs. This may be an advantage in the day-to-day administration of the Netarkivet repository. Of course, the network load will be smaller too, but note that when a failure does occur (when two disks fail on the same PC), the content of 6 disks (1,5TB) must now be transferred. This will take longer time than just transferring the contents of two disks (as in our standard model), and longer repair time raises the risk of data being lost (see [Crespo00ext]).

The central question, however, is to the effect of RAID on the MTTF of the complete repository. We created an alternative model to represent that design, but we do not have the MTTF of the alternative model on realistic data amounts (in the 20-40TB range) because the computations took too long. We shall need to optimize our simulation system greatly to be able to estimate that MTTF. We modified our system to stop a simulation when 200 years had passed. With this modification, we observed that out of 250 simulations of the alternative model, only one lost data before 200 years. This should be compared with the MTTF of our standard model, which was 144 years. If the amount of ingested data was raised from 20TB to 40TB, still only 9 out of 250 simulations had data loss before 200 years.

We investigated the MTTF of the complete repository on very large data amounts in the setup where the PCs were RAIDed; see Figure 10. Notice the scale on the X-axis. When RAIDing the PCs, the combined repository has a MTTF of 44 years - when an amount 500TB, half a petabyte, is ingested. With this amount of data, our standard model has a MTTF of less than 2 years.

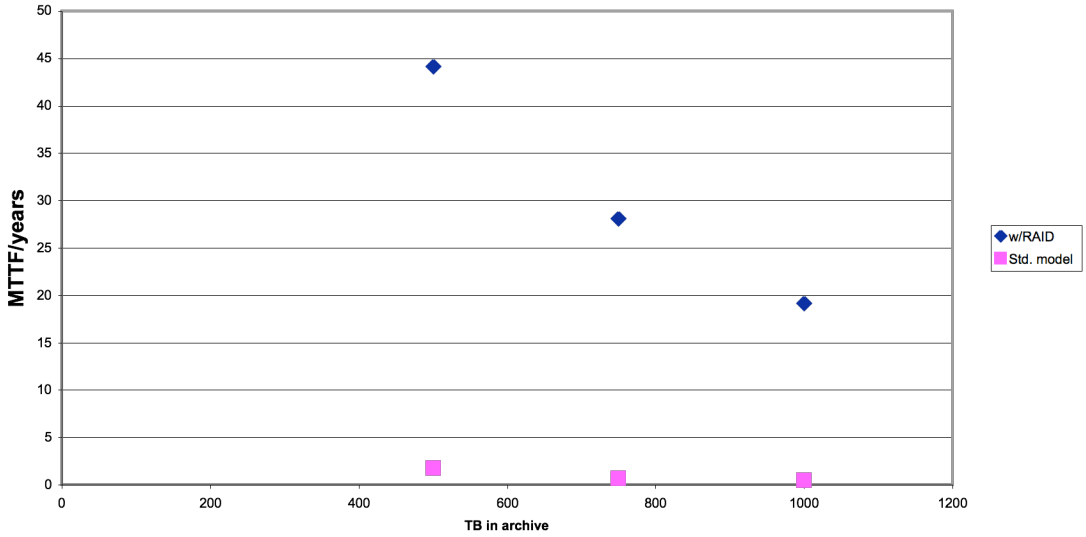

Figure 10:The mean time to failure of the Netarkivet repository if each PC in the KB cluster used RAID.

5. Conclusion and future work

In this paper, we have described the archival, digital repository that was installed to preserve bit-by-bit the data generated by harvesting the Danish Internet. We also described a probabilistic model of that repository, and summarized the results of our computer simulations based on the model. In this section we list our conclusions regarding our own repository, regarding repositories in general, and regarding our method of evaluation. We also mention some of our intended future work on the subject.

5.1 Conclusions regarding the repository of Netarkivet

The question, which more than anything else motivated our study, was whether the MTTF of the repository that we created is reasonably high. Our model is built on many assumptions, but we dare to conclude as much: Within a time span of several decades, we should rather spend our efforts improving the repository's resilience towards other factors than the ones studied in our simulations. For instance, if we do not put more effort into the logical preservation of our materials, then on a 20-year time scale data loss due to format obsolescence seems far more probable than data loss caused by several disks/tapes failing within a very brief period of time. And in the same line of reasoning: While investing in RAID on the KB bitarchive machines will have a big impact on the MTTF factors that we studied in this paper, our money are probably better invested in improving our logical preservation, among other things.

5.2 Conclusions regarding digital repositories in general

In the experiment summarized in Section 4.2 we were able to verify our intuition that an archival, digital repository like our own can ingest an accelerating amount of data each year without suffering from decreasing MTTF or increasing costs. To achieve this, the repository needs to keep up with the ever-improving storage technology. (And it is a premise that the improvements in storage technology can be equalled to the acceleration in the size of ingested data). There is a converse implication of that conclusion: An archival, digital repository must keep up with the latest storage technology if it ingests ever-increasing amounts of data. If it does not, it will suffer an ever-increasing risk of data loss or accelerating costs. Of course, our experiment only concludes this for our specific model of a repository, so we cannot rule out the existence of some clever repository design for which our conclusion does not apply. We conjecture that such a design is not possible, but would be very interested in suggestions of candidates.

5.3 Conclusions regaring the evaluation method

We chose to evaluate our own design using the method of [Crespo00ext]. As the above conclusions indicate, we feel that the method has provided us with valuable information about our repository. We fully recommend other repository designers to investigate their ideas through simulations like our own, and our experience was that the effort we put into setting up and running the experiments was far, far less than the effort it took to arrive at a repository design in the first place.

5.4 Future work

In continuation of the work presented in this paper, the author intends to:

- Improve our simulation software, making it simpler to define new models and making the simulator more efficient.
- Investigate more comprehensive models of our repository, thus removing some of the more annoying assumptions described in Section 3.1.
- Extend the simulator to compute more than MTTF. In particular, we would like the system to compute the costs associated with a given design, given some basic cost parameters.

5.5 Acknowledgements

The author gives his most sincere thanks to Ulla B. Kejser, Jakob G. Simonsen and Steen S. Christensen for their many useful comments and suggestions. I would also like to thank the anonymous referees for their comments.

6. References

[Arc] ARC file format, http://www.archive.org/web/researcher/ArcFileFormat.php

[Crespo00] Arturo Crespo, Hector Garcia-Molina: "Modeling Archival Repositories", in Proceedings of the Fourth European Conference on Digital Libraries (ECDL). September 2000, http://www-db.stanford.edu/~crespo/publications/ArchSim.pdf

[Crespo00ext] Arturo Crespo, Hector Garcia-Molina: "Modeling Archival Repositories", extended version of the above, http://www-db.stanford.edu/~crespo/publications/ArchSimFull.pdf

[Crespo03] Arturo Crespo: "Archival Repositories for Digital Libraries", Doctoral Dissertation, Stanford University, March 2003, http://www-db.stanford.edu/~crespo/publications/thesis.pdf

[Dpc05] Jim Linden, Sean Martin, Richard Masters, Roderic Parker: "The Large-scale Archival Storage of Digital Objects", Technology Watch Report 04-03, Digital Preservation Coalition, February 2005, http://www.dpconline.org/docs/dpctw04-03.pdf

[Heritrix04] Gordon Mohr, Michele Kimpton, Micheal Stack, Igor Ranitovic: " Introduction to Heritrix, an archival quality web crawler", Proceedings of the International Web Archiving Workshop '04, 2004, http://www.iwaw.net/04/proceedings.php?f=Mohr

[Iipc] International Internet Preservation Consortium, http://www.netpreserve.org/

[Md5] Message-Digest algorithm 5, see http://en.wikipedia.org/wiki/MD5

[Oais] Reference Model for an Open Archival Information System, Consultative Comittee for Space Data Systems, http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html