Modeling Archival Repositories for Digital Libraries*

Arturo Crespo, Hector Garcia-Molina
Computer Science Department
Stanford University
{crespo,hector}@db.stanford.edu

ABSTRACT
This paper studies the *archival problem*: how a digital library can preserve electronic documents over long periods of time. We analyze how an archival repository can fail and we present different strategies that help solve the problem. We introduce *ArchSim*, a simulation tool that for evaluating an implementation of an archival repository system and compare options such as different disk reliabilities, error detection and correction algorithms, preventive maintenance, etc. We use ArchSim to analyze a case study of an Archival Repository for Computer Science Technical Reports.

KEYWORDS: digital archiving, digital preservation, archival repository, models for archival repositories, performance of archival repositories, simulation of archival repositories.

1 Introduction

A continual threat to digital libraries is the loss of documents. Digital information can be lost not just through magnetic decay in storage devices, but also because of format and device obsolescence. This problem will only get worse as more and more information is provided only in digital form. The solution is an *Archival Repository* (AR), a system capable of storing and preserving digital objects (e.g., movies, technical reports) as technologies and organizations evolve [3]. An AR must preserve not only the bits, but also the "meaning" of its documents [9]. For instance, to preserve a "postscript" technical report, the AR needs to maintain the postscript bits, metadata indicating that this document is postscript, a program that can interpret and render postscript, and an environment that can execute the rendering program.

One of the main challenges in designing an archival repository is how to configure the repository to achieve some target "preservation guarantee" while minimizing the cost and effort involved in running the repository. For example, the AR designer may have to decide how may sites to use, what types of disks or tape units to use, what and how many formats to store documents in, how frequently to check existing documents for errors, what strategy to use for error recovery, how often to migrate documents to a more modern format, and so on. Each AR configuration leads to different levels of assurance, e.g., on the average a document will not be lost for 1000 years, or in 1000 years we expect to still have access to 99% of our documents. Each configuration has an associated cost, e.g., disk hardware involved, computer cycles used to check for errors, or staff running each site.

The number of options and choices is daunting, and the AR designer has few good tools to help. The traditional fault tolerance models and techniques, of the type used to evaluate hardware, are a helpful starting point, but they do not capture the unique complexities of ARs. For example, traditional models may have difficulty capturing different document loss scenarios (e.g., missing interpreter, missing bits, missing metadata) and they frequently assume failure distributions (e.g., exponential) that are too simplistic.

In this paper we present a powerful modeling and simulation tool, *ArchSim*, for helping in AR design. ArchSim can model important details, such as multiple formats, preventive maintenance, and realistic failure distribution functions. ArchSim is capable of evaluating a large number of components over very long time periods. ArchSim uses specialized techniques in order to run comprehensive simulations in a time frame that allows the exploration and testing of different policies.

Of course, no model can be absolutely complete: there is an intrinsic tradeoff between how detailed the model is and the complexity (even feasibility) of its study. In our case, we have chosen to ignore (at least in this initial study) information loss due to format conversion (migration to a new format is always successful and does not introduce any loss). We also do not model partial failures (a failure where we can salvage part, but not all, of the information in a device). As we will see, we also rely on an "expert" that can provide failure distributions for the components of the systems. For instance, if format obsolescence is an issue, an expert needs to give us the probabilities of different format lifetimes.

In summary, our contributions are:

- A comprehensive model for an AR, including options for the most common recovery and preventive maintenance techniques (Sections 2, 3 and 4).

*This work was partially supported by NSF.

- A powerful simulation tool, ArchSim, for evaluating ARs and for studying available archival strategies (Section 5).
- A detailed case study for a hypothetical Technical Report repository operated between two universities. Through this case study, we evaluate AR factors such as disk reliability, handling of format failures, and preventive maintenance.

2 Archival Problems and Solutions

As a first step in modeling and evaluating archival repositories (ARs), it is important to understand how information can be lost, and what techniques can reduce the likelihood of loss. Because of space limitations, we limit ourselves to a brief review of failure sources and avoidance techniques. The expanded version of this paper [8] contains the full review.

2.1 Sources of Failures

An AR fails to meet its guarantee when it loses information. Such a loss may be caused by a variety of undesired events, such as the failure of a disk, on an operator error. A document is lost if the bits that represent it are lost, and also if the necessary components that give meaning to those bits are lost. We define the *components* of a document to be all the resources that are needed to support access to the document, e.g., the document bits, the disk that stores the bits, and the viewer that interprets the bits.

An undesired event does not necessarily cause information loss. For instance, if the AR keeps two copies of a document, and the disk holding one of the copies fails, then the document is not lost. It would take a second undesired event affecting the second copy to cause information loss. A document is *damaged* if a copy or instance of one of its components is corrupted or lost. Damaged documents should be *repaired* by the system to protect them from further failures.

The most common undesired events that may lead to the loss of a document can be broken down into the following categories. (We do not consider transient failures, e.g., a power failure, that do not lead to a permanent loss.)

Media decay and failure: Example: disk magnetic decay; a worn out tape breaks when it is being read.

Component Obsolescence: Example: we do not have a machine that can read a tape, even if the media is still readable.

Human and Software Errors: Example: a person or a program deletes a document; a program improperly modifies the files and data structures that represent the document. (Reference [13] suggests that humans and software are the most serious sources of failures in computer systems.)

External Events: Examples: fires, earthquakes and wars. Such events may cause several AR components to fail simultaneously. For example, a flood can destroy a collection of disks at one site. If a document was replicated using those disks, all copies will be lost.

2.2 Component Failure Avoidance Techniques

There are two well-known ways to avoid an AR failure: We can either reduce the probability of component faults, or we can design the AR so those faults do not result in document loss. In this subsection, we review techniques in the first category, while the next subsection will cover the second category. We organize the presentation in this subsection using the taxonomy of faults presented in Section 2.1.

Media decay and failure: To reduce the likelihood of media failure, we can store our document on very reliable media (e.g., use CDs instead of tapes). We can also ensure that the media is maintained in the best conditions possible. For example, by placing tapes in a low-temperature/low-humidity environment, we may increase their life by an order of magnitude [2].

Component Obsolescence: Reducing component obsolescence is hard, as it requires an accurate prediction of what operating systems, document formats, and devices will be used in the future. Still, use of standards, self-contained media (i.e., media that includes its own reader) [14], and equipment preservation may help.

Human/Software Errors: Good coding techniques can reduce the likelihood of these failures, e.g. define interfaces that minimize the amount of damage that can be done. In addition, program validation and operator training can help.

External Events: Damage from external or environmental events can be reduced by fire-proof walls, earthquake resistant buildings, and so on.

2.3 System-Level Techniques

Migration: Migration involves replacing a document component by a new, safer one. For example, suppose that "Postscript" readers are becoming obsolete, being replaced by new "Postscript II" ones. Then we may decide to migrate Postscript documents into the new format, before Postscript readers become unavailable. Migration is particularly effective for storage devices. However, migrating to new formats is more challenging because some information may be lost in the transformation.

Replication: Replication makes copies or creates new instances of needed components, to ensure the long term survival of the component. Replication is used by many archival system including the Computing Research Repository [12], the Archival Intermemory Project [10, 6], and the Stanford Archival Vault [4].

Emulation: Emulation involves re-creating all the components needed to access a document on a new platform [17]. Emulation can be done at several levels: hardware, the operating system, software, or format.

Fast Failure Detection and Repair: With most system fault tolerance techniques, we need to check periodically for failed documents. Fast failure detection and repair yields improved reliability. For example, if one of two component copies have failed, the sooner we detect the problem and generate a new copy, the more protection we get against a second fault.

3 Architecture of an Archival Repository

Our goal in this section is to identify the elements of a typical archival repository (AR), so we can model each element and study how it impacts reliability. A typical AR stores documents in a *data store* that can fail. The AR can still achieve long-term survivability by enhancing the data store with an *archival system* (AS) that implements some of the techniques of Section 2.3. We present our AR model in Figure 1. The figure shows the AS modules (in solid-line boxes), the non-fault-tolerant store (in a dashed-line box), and the archival documents. The arrows represent the runtime interactions between the elements.

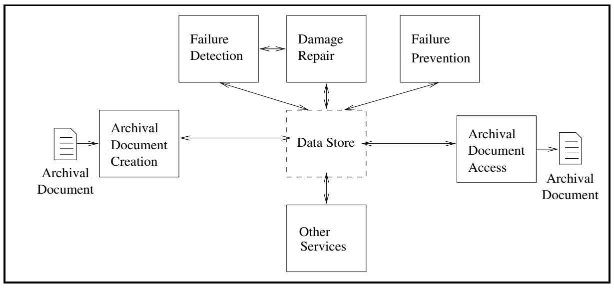

Figure 1: Archival Repository Model

3.1 Archival Documents

In our architecture, an archival document embodies information. An archival document cannot be just a bag of bits, but it must also include all the components necessary to transform the bits into a human comprehensible form.

An archival document is an abstract entity. The connection between document and access to it is achieved through *materializations.* A materialization is the set of all the components necessary to provide some sort of human access to a document. For example, a materialization may include the bits, disks, and format interpreters necessary to display a technical report. The same technical report may be accessible through a different materialization, that may include a different format interpreter to print the technical report.

We illustrate materializations in Figure 2. In the figure, there are two archival documents. Each of those documents has two different materializations. For example, Materialization 1 requires the following components to be available: File 1, Site A, Disk 1, and a ASCII printer. Incidentally, notice that File 1 is stored on Disk 1, which in turn is at Site A. Such component interdependencies will be discussed later, when we model materialization failures.

The AR is able to preserve documents by preserving the materializations and their components. In this paper, we treat the documents as "black boxes." We do not attempt to take advantage of document structure (e.g., chapters in a book).

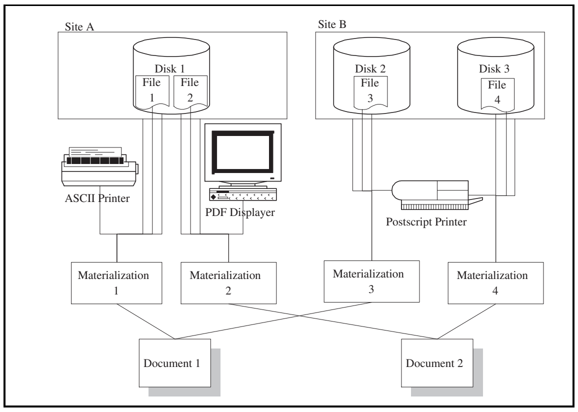

Figure 2: Materializations

3.2 Architecture of the Non-Fault-Tolerance Store

The *Store* encompasses the set of components, such as sites, disks, or format interpreters that make materializations accessible. Because the store is not fault tolerant, materializations may be lost. A materialization is considered lost when *any* of its components has failed. If *all* of the materializations of a document are lost, then the document is considered lost.

To create a materialization, first we must ensure that the necessary components exist in the store. Then we create a record that links together the components as a materialization. For example, say we want to create a document materialization that requires the bits in file "doc.ps", which are located in $disk_2$ in $site_1$ and requires the postscript interpreter. Any components that do not exist already (e.g., the file doc.ps) are created. In some cases, "creating" a component may require a physical action, e.g., adding a new printer or disk to the store. Once the components exist, a metadata record containing references to the components is created (e.g., $\langle site_1, doc.ps, disk_2, postscript\rangle$). This record serves as the identifier for the materialization.

A document metadata record includes the records for all available materializations. Note that a document is not accessible if its metadata record is corrupted or lost. Therefore, the record must be one of the required components for any materialization.

3.3 Architecture of the Archival System

The AS provides fault tolerance by managing multiple materializations for each document. The AS monitors these materializations, and when a failure is detected, attempts to repair them. The AS can improve fault tolerance further by taking preventive actions to avoid failures. The AS provides the user the ability to create and retrieve archival documents. It also provides miscellaneous services such as indexing, security, and document retirement, among others. When a user requests a document, the AS uses its metadata to find all the available materializations of that document, selects one and returns it to the user. In this section, we describe the six modules that make up at AS (see Figure 1).

The *Archival Document Creation module* (ADC) generates new documents, implementing policies on the number and types of materializations that are needed. For example, say an administrator has decided that documents should be materialized as illustrated by Document 1 in Figure 2. Then, for each new document, the ADC will create (on the data store) the appropriate "File 1" and "File 2," the document metadata record, and will check that the other components exist. The main objective of this paper is to provide a framework that allows a system administrator to choose the best materialization policies to achieve a desired level of reliability.

The *Archival Document Access module* (ADA) services request for documents. Basically, the module translates the request for a document into a request for the appropriate components of one of the document materializations.

The *Failure detection module* (FD) scans the store looking for damaged or lost materializations. When a damaged materialization is found, the failure detection module informs the Damage Repair module (described below) about the problem.

The *Damage Repair module* (DR) attempts to repair damaged documents. There are many strategies to repair a damaged document, as discussed in Section 2.3. The input of the DR module is a signal from the FD module.

The *Failure Prevention module* (FP) scans the store and takes preventive measures so materializations are less likely to be damaged. For example, the FP module may copy components that are stored on a disk that is close to the end of its expected life, into a newer disk.

Finally, the *Other Services module* (OS) provides miscellaneous services such as indexing, security, and document retirement. Retiring a document involves removing from the store any components that are no longer needed, even by other materializations.

4 Failure and Recovery Modeling

In this section, we will model the failure and recovery characteristics of an AR, based on the architecture presented in the previous section. First, we will explore how to model a non-fault-tolerance store, and then the archival system (AS). Later, we will combine these two models into an archival document model.

4.1 Modeling a Non-Fault-Tolerance Store

To model the failure characteristics of a store, we start with an abstract representation of materializations and components. We model a materialization as an n-tuple $\langle mat_{id},\ doc_{id},\ comp_1, ...comp_n\rangle$; where mat_{id} is the materialization identifier, doc_{id} is the document identifier, and $comp_1 ...comp_n$ are the components required to provide the required document access. The identifiers mat_{id} and doc_{id} together, form a unique id for the materialization. For example: $\langle M1,\ TR1233,\ doc.ps,\ site_1,\ disk_2,\ postscript\rangle$ means that the materialization $M1$ contains the document identified by $TR1233$ that needs the bits in file $doc.ps$, $disk_2$, $site_1$ and the postscript interpreter in order to be readable. A document can have more than one materialization. For example, Technical Report 1233 can also have the materialization $\langle M2,\ TR1233,\ doc.ps,\ site_1,\ disk_3,\ postscript\rangle$, which would be a copy of M_1 but on a different disk ($disk_3$).

We model components by a tuple $\langle component_{id},\ type\rangle$, where $component_{id}$ is a unique identifier for the component instance and $type$ is the class (e.g., file, disk, interpreter) to which the instance belongs.

To further model components, we need to describe:

- How many component instances and types are present in the system: this is, how many disks, formats, etc., are available.
- Failure distribution of each component type. Many components have two different failure distributions, one during archival and another during access. For example, a tape is more likely to fail when it is being manipulated and mounted on a reader than when it is stored. Therefore, each component will have two failure distributions: during archival (i.e., time to next failure when the component is not used) and during access. For some components, such as disks or sites, the access and archival distributions will be the same; but for other components, such as tapes or diskettes, these distributions can be very different.
- Time distribution for performing a component check. This distribution describes how long it takes to discover a failure (or to determine that a component is good), from the time the check process starts. For example, consider checking a tape. This may involve getting the tape from the shelf, mounting the tape, and scanning the tape for errors.
- Time distribution for repairing a component failure. This distribution describes how long it takes to repair a component. This distribution may be deterministic (if the component can be repaired in a fixed amount of time). Repair time may be "infinite" if the component cannot be fixed.

In addition, there is an important interdependency between components. Specifically, the failure of one component may cause the failure of another component. For example, if a site fails (e.g., because it was destroyed by a fire), then all the disks at the site will also fail. As pointed out earlier, we are only taking into account permanent failures; transient failures (e.g., the site was temporarily disconnected from the network) are ignored. This failure dependency is captured by a directed graph. For example, an arrow between "Site A" and "Disk 1" in the interdependency graph means that if "Site A" fails, then "Disk 1" will also fail.

We close this subsection with two comments. First, we do not claim that the model presented for the store is complete. For instance, we have not included policies for handling concurrent access. There is always a tradeoff between complexity of the model and our ability to analyze it. We believe that our model strikes a good balance in this respect, and captures the essential features of a store. Second, the reliability predictions we make are only valid for the *current* configuration of the repository. Over time, the repository will change (e.g., as new devices are introduced), so we may need to change our repository model. As the model changes, we may need to revisit our predictions.

4.2 Modeling an Archival System

In this section, we describe how to model the behavior of the modules of the archival system. We do not include failure distributions for these modules as we are assuming that the AS itself does not fail. We recognize that this is a strong assumption, but in this paper, we have chosen to concentrate on the failure of components, instead of on the failure of the system that provides fault-tolerance. How to develop error-tolerant robust software design have been study in [16].

Because of space constraints, we cannot describe each module separately. In general, we model the input of the modules with probability distribution functions and their behavior by algorithms. For example, consider the document creation (ADC) module. Its input distributions tell us how frequently requests for document creation arrive, how many materializations each new document will have, and which components will be selected to participate in a given materialization. The algorithms for the failure detection (FD) module spell out what policies are implemented, e.g., if all components are checked on a regular basis or not.

The probability distributions that drive the model can be obtained in different ways. If we have data from a real system, we can use the data directly (trace driven), or we can define an empirical distribution, or we can fit the data onto a theoretical distribution [1]. If we do not have real data, we need to choose a theoretical distribution that matches our intuition. A sensible distribution to choose (when requests are generated independently) is a Poisson distribution for event inter-arrival times [5].

We summarize the model parameters in Figure 3. The figure is divided in three parts. At the top are the parameters that describe the AR: the number of components and their types, and the failure dependency graph. Then, we list all the distributions needed for the model with the units being modeled in parenthesis. Finally, we list all the policies and algorithms that must be defined to model the archival system.

4.3 Modeling Archival Documents

In this section, we combine the models for the data store and the AS, in order to describe the life of an archival document. In Figure 4, we depict our model for the life of an archival document. The life of a document starts when its materializations are created by the ADC module and handed to the store. The creation of a document may not be an instantaneous process. For example, if long-term survival is achieved by keeping multiple copies, the document is not considered archived until all the copies are generated. Once the ADC module has taken all the actions that ensures the long-term

- AR Description
  - Number of components and types
  - Failure dependency graph
- Distributions
  - For each component type:
    - Failure distribution during access (time)
    - Failure distribution during archival (time)
    - Failure detection distribution (time)
    - Repair distribution (time)
  - Document creation distribution (time)
  - Document access distribution (time)
  - Access duration distribution (time)
  - Document selection distribution (document)
- Policies
  - Document Creation policy
  - Document to materializations policy
  - Failure detection algorithm
  - Damage Repair algorithm
  - Failure prevention algorithm

Figure 3: Archival Repository Model Parameters

survival of the document, then the document has full protection, and we say that the document is in the *Archived* state.

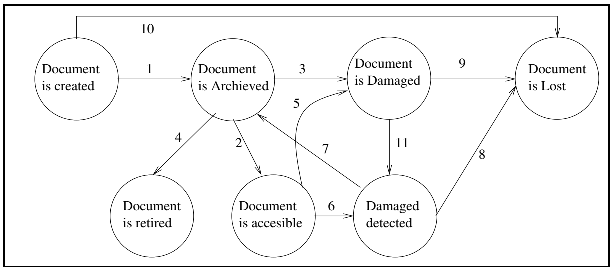

Figure 4: Archival Document Model

When any of the document component fails (based on the distributions in Figure 3), the document is considered to be in the *Damaged* state and becomes temporarily unprotected. For example, if we keep two copies of a document and one of the copies is lost, then the document would be damaged. As explained earlier, the AS will not know that a document is damaged until the FD module detects the failure. When the failure is detected, the document goes to the *Damaged Detected* state.

When damage to a document is detected (by the policies summarized in Figure 3), the AS starts actions to restore the document and, hopefully, return it to the Archived state. For example, if the document is damaged because one of its copies was lost, the repository can just replace the damaged copy by creating a fresh one from one of the good copies. However, if the repair is not successful, then the document may be *Lost*. This later state is the one that we want to avoid in an archival system.

We also distinguish two additional states: Accessible and Retired. When a document is in the *Accessible* state, it can be accessed (e.g., read, printed) by users, which is not the case for the Archived state. For example, if some the document's components are stored on a tape which is kept in a safe, we need to take the tape out and mount it in a reading device to make it accessible. When the tape is stored the document is in the Archived state; when the tape is mounted, it is in the Accessible state. As we discussed in the previous section, when making the document accessible, in general, we are increasing the chances of damaging the document; so the probability of transition 7 is, in general, greater than the probability of transition 5.

The Retired state allows users to mark the document that are not needed anymore. In this case, the document is retired from the archival system and the system does not provide any survivability guarantees. It is important to note that retiring a document may eventually result in removing all materializations from the store. This action is different than taking a document "out of circulation," in which case the document is not longer available to regular users, but it is still preserved for historical reasons.

5 ArchSim: A Simulation Tool for Archival Repositories

To evaluate a possible AR configuration, we need to predict how well it protects documents. This prediction can sometimes be done analytically, but as the AR gets more complex, an analytical solution is impractical (and sometimes impossible). Instead, we rely on a specialized simulation engine for archival repositories: ArchSim. We start this section by discussing the specific challenges confronted when simulating an AR. Then we describe ArchSim and its libraries.

5.1 Challenges in Simulating an Archival Repository

ArchSim builds upon existing simulation techniques for fault-tolerant systems. However, the unique characteristics of archival repositories make their simulation challenging:

- *Time Span:* The life of an archival system is measured in hundreds, perhaps thousands of years. This means that simulation runs will be extremely long, so special precautions must be taken to make the simulation very efficient. Furthermore, given these long periods, failure distributions must take into account component "wear-out." (A component is more likely to fail after 50 years that it is when new.) Simple failure distributions (e.g., exponentially distributed time between failures) are frequently used in fault-tolerant studies, but they cannot be used here since they do not capture wear out.
- *Repairs:* In an archival system we cannot in general assume that damaged components can always be replaced by new identical components (another common assumption when studying fault-tolerant systems). For example, after say 100 years, it may be impossible or undesirable to replace a disk with one having the same failure characteristics.
- *Component models:* Component models are fairly rich, compounding the number of states that must be considered. For instance, as we have discussed, a file is not simply correct or corrupted. Instead, it can be corrupted but the error undetected, it can be correct but not accessible for reads, and so on. The failure models in each of these states may be different, e.g., a file is more likely to be lost when being read.
- *Sources of failures:* A document can be lost for many reasons, e.g., a disk fails or a format becomes obsolete. Each of these failures has very different models and probability distributions. The approach of finding the "weak link" and assuming that all other factors can be ignored is not appropriate for ARs.
- *Number of Components:* An AR needs to deal with a large number of components and materializations. The challenge of simulating large number of objects has been studied extensively [11, 15] and ArchSim uses those results.

5.2 The Simulation Engine: ArchSim

ArchSim receives as input an AR model, a stop condition (stop when the first document is lost or when all documents are lost) and a simulation time unit (minutes, hours, days, etc.). ArchSim outputs the mean time to failure (mean time to stop condition), plus a confidence interval for this time. We are currently considering other output metrics, e.g., the fraction of the documents that are available after some fixed amount of time. However, these other metrics are not used in our case study (Section 6).

For defining the AR model, each distribution and policy is implemented as a Java object, so they can be as general as necessary. For example, for component repair, the corresponding Java module can simply use a probability distribution (perhaps one of the library functions described below) to generate the expected repair time. However, that module can easily be replaced by one that first decides if the component can be repaired (using one probability distribution), and then for each cases generates a completion time (when the component is repaired or the repair is declared unsuccessful).

5.3 Library of Failure Distributions

ArchSim makes available a library of pre-defined failure distributions, that can be used to describe AR components. The distributions in the library are: bathtub, infant mortality, historical survival, uniform, and deterministic. In Figure 5(a)-(e) we sketch generic versions of these probability distributions. For instance, in the bathtub distribution in (a), the instantaneous probability of failure early in the component's life (left on the horizontal axis) and late in its life (to the right) are higher than during the middle years. With the deterministic distribution (e), the component fails at a fixed time, where a spike is shown. (The area under these curves, from time 0 to t, represents the probability the component will fail by time t.) In the extended version of this paper [8], we give a formal definition of each distribution.

5.4 ArchSim's Implementation

ArchSim follows the structure of a traditional simulation tool. Each module of the AR model can register future events in a timeline. For example, when a disk is created, the simulation

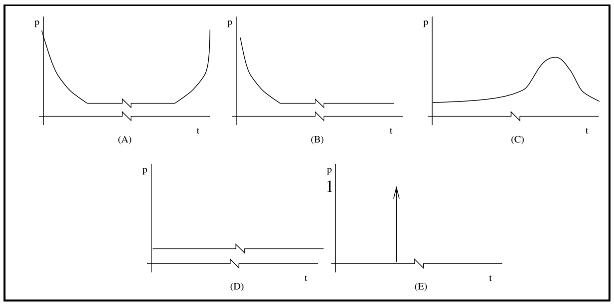

Figure 5: Possible Failure Functions

uses the disk failure distribution to compute when the disk will fail; then, it registers the future failure event in the timeline. The simulation engine advances time by calling the module that registered the first event. This module may change the state of the repository and register more events in the timeline. After the module returns, the simulation advances to the next event, in chronological order. The user can choose between two end conditions for the simulation: the simulation can stop after the first document is lost or after all the documents are lost.

ArchSim needs to be very flexible and efficient to meet the challenges of simulating an archival repository. Flexibility is needed to model very different archival conditions and implementations. Speed is needed to cope with many materializations, components, and events. Additionally, each simulation needs to be run many times in order to obtain narrow confidence intervals.

In an AR simulation, many events are inconsequential. For example, suppose that the detection module schedules periodic detection events. If the detection event finds a fault (i.e., there was a failure event before the detection event), then the module starts a component repair; if no failure is detected, then the module does nothing. If a repair module checks a component with mean time to failure (*MTTF*) of 20 years, every 15 days, then, in average, 486 events ($20 * 365/15$) will be fired and the repair module will just return without doing anything; only in the event 487, when a failure of the component has happened, will the repair module perform an action. Given the large number of components and modules that may be part of the model, this large number of inconsequential events represents a significant overhead. To avoid this overhead and to improve efficiency, modules are allowed to register *conditional* events in the timeline. These events will only happen if some other event happened before them. By using conditional events, we can condition the firing of the detection event only if a failure event on an specific component happened before it. A conditional event is not registered directly in the timeline. Instead, it is registered in an index that is part of its triggering event. For example, if event B is conditional to the occurrence of event A, we will put B in the index of A. When a new event A is schedule in the timeline, we look in the index and find that B is conditional to it, so at that point we also schedule B.

To reduce the number of events further, ArchSim also modifies failure distributions. For example, when modeling preventive maintenance, a large number of "replace component" events are generated. We can eliminate all those events, by modifying the failure distribution. Specifically, the original failure distribution is used to generate the time of the next failure of the component. If this time is higher than the prescribed preventive maintenance period, we ignore this time, and we generate a new failure time, gain using the original distribution. We repeat this process until a failure time is lower than the preventive maintenance period. The new distribution then returns the number of iterations minus one, times the PM period, plus the last failure time.

Another challenge for ArchSim is how to deal with a large number of materializations and components. This is done by scheduling only the *next* failure for each component type and associating a trigger with that event. When the failure event happens, the trigger is activated. The trigger computes when the *next* failure of a member of that component type will occur, and adds it to the timeline. Obviously, this approach is only beneficial if we have many components of the same type, which is a reasonable scenario for an AR. In the case when we have N different distributions for N components, this technique does not improve the simulation time, but it does not increase it.

6 Case Study: MIT/Stanford TR Repository

In this section, we use ArchSim to answer some design questions for an hypothetical MIT/Stanford Technical Report Archival Repository. The AR follows loosely the Stanford Archival Vault (SAV) design [7] and implementation [4]. (We actually considered creating such a repository some years ago, when both institutions were participating in the DARPA sponsored CSTR Project.) In this case study, MIT and Stanford preserve their Computer Science Technical reports by replicating the reports at both universities. In this case study we will have to make many assumptions. Our goal here is *not* make any specific predictions, but rather to illustrate the types of evaluations that ArchSim can support, the types of decisions that must be made to model an AR, and the types of comparisons than can be made to support rational decisions among alternatives.

We will assume that the collection has 200,000 documents and that each document is stored in one or more of four available formats. The repository will have two types of components: storage devices (disks) and format interpreters. To ensure preservation, the AR maintains four materializations of each technical report; two materializations at Stanford and two at MIT. Materializations are created by choosing two formats out of the four available formats. Two of the materializations will be in one of the chosen format, while the other two will be in the other. Then, at each site, we place two materializations that are in different formats in two different disks. Figure 6 illustrates the arrangement of the technical reports in this system.

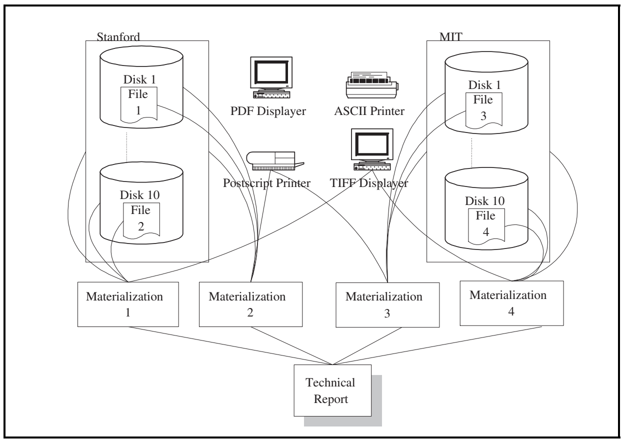

Figure 6: MIT/Stanford CSTR Scenario

Disks, formats, and sites have uniform failure distributions with parameters $1/\phi_{sto}, 1/\phi_{form}, 1/\phi_{site}$. As our base values, we are assuming ϕ_{sto}, ϕ_{form}, and ϕ_{site}, the mean time to failure (MTTF) for disks, formats and sites, to be 3, 20, and 45 years respectively.

These values are our best guess for a typical archival system. We chose 3 years as the disk MTTF, as this is the normal period under which a hard drive is under manufacturer guarantee. We chose 20 years for format MTTF as we theorize that it will take that long for a well-known format to be replaced by a new format and for all displayers and transformers for the old format to be lost. We chose 45 years for the site MTTF as we assign a 50% probability to the event of loosing the Stanford site due to a high intensity earthquake in the San Francisco Bay Area (which is predicted to happen in a 45 year period).

The archival system checks for faults periodically. When a fault is detected in one of the storage devices, the bad device is retired, and a new device is set online. Then, the system regenerates the bad device by making a copy of the lost materializations from the other site. Similarly, when a format becomes obsolete, a new format is selected and a new set of materializations (transformed from a non-obsolete format) is created in the new format. In addition, in case of site failure, the site is recreated from the other site. If all sites, formats and devices that support all the materializations of a technical report are lost, then the technical report is lost and the simulation stops. Disks, formats, and sites are checked and repaired (if needed) every ρ_{sto}, ρ_{form}, and ρ_{site} days. As our base values, we are assuming ρ_{sto}, ρ_{form}, and ρ_{site} to be 60, 60, and 7 days.

To justify these values, we need to describe what is involved in the detection and repair of component failures. Detecting a failure in a disk involves scanning the whole disk and checking for lost data. When we find lost data, we need to order a new disk and then copy all the data that was in the damaged

Parameter	*Symbol*	*value*
Number of disks	n_{sto}	100 per site
Number of formats	n_{form}	4
Number of documents	num_{doc}	200,000
Mean Time to Disk Failure	ϕ_{sto}	3 years
Mean Time to Format Failure	ϕ_{form}	20 years
Mean Time to Site Failure	ϕ_{site}	45 years
Disk Failure Detection/Repair time	ρ_{sto}	60 days
Format Failure Detection/Repair time	ρ_{form}	60 days
Site Failure Detection/Repair time	ρ_{site}	7 days

Figure 7: Base values

disk from other sources into the new disk. Assuming that the repair time is 60 days means that we need to dedicate only 3% of the disk bandwidth to scan all materializations in order to detect failures. We did not chose a quicker repair because the scanning overhead would be too high in our opinion. For example, 25% of the disk bandwidth is required to detect failures in 7 days.

For formats, the detection/repair times imply that we are able to realize that a format is obsolete and that we can create a new copy of the document from a non-obsolete format within a 60 day period. In the case of site failures, we are assuming that the detection/repair time for the site is much lower than for formats and disk. The detection itself should be rather fast in this case (the entire site is down), and the 7 days could be the time it takes to find a backup site to take over.

In this case study, we are assuming that failures are total. This is, we cannot partially repair a component and salvage some of the materializations. The failure distribution during access will be assumed to be the same as the failure distribution during archival. The simulation parameters are summarized in Appendix I and the base values for our simulation are in Figure 7.

In our first experiment, we evaluate the effect of the failure MTTF and repair times of storage devices on the system MTTF. In Figure 8, we show the system MTTF for different disk MTTFs, given a detection/repair time of 60 days. To single-out the influence of storage device failures, we are assuming in this experiment that formats and sites never fail. The dotted lines in the figure represent the 99% confidence interval for the simulation, while the solid line is the average of all the simulation runs. As expected, the system MTTF increases when the disk MTTF increases (when the disk failure rate decreases). The exponential shape of the curve is the result of the constant repair time. As we keep increasing the disk MTTF, it is much more improbable that another device will also fail before 60 days have passed. This graph allows us to select a good storage device for a target system MTTF. If the library targets a 10-year MTTF, then a disk with a failure MTTF of 3 years will suffice. However, if the library requires a MTTF of 100 years, then we will need disks with a MTTF of about 6 years. Most manufacturers guarantee their disks for

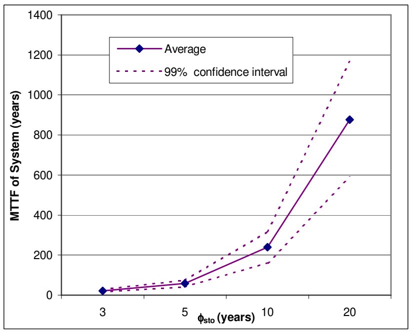

Figure 8: MTTF vs. ϕ_{sto} with ρ_{sto} of 60 days

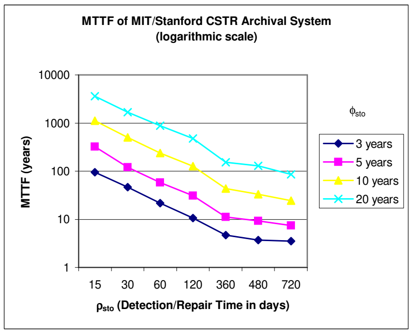

Figure 9: System MTTF (logarithmic scale)

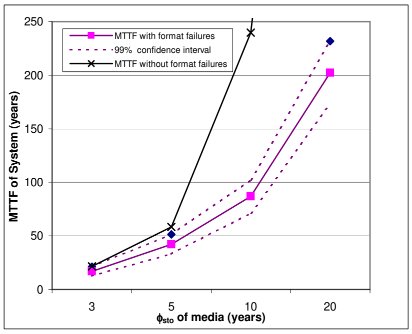

Figure 10: MTTF ϕ_{form}=20 years, ρ_{sto}=ρ_{form}=60 days

three years and very few guarantee them beyond five years. Therefore, a 6-year disk MTTF requirement will be hard (or very expensive) to meet. Nevertheless, we can still achieve our target system MTTF by changing other parameters in our system, as we will see in the next experiment.

We now evaluate the sensitivity of the system MTTF to the repair time (ρ_{sto}). In Figure 9, we show the MTTF of the AR for different disk MTTF (ϕ_{sto}) values, and for different expected detection/repair times. (Note that the graph has a logarithmic scale. Confidence intervals are not shown to avoid clutter.) First, let us concentrate in the curve for a ϕ_{sto} of 3 years. As expected, the MTTF of the system decreases when the ρ_{sto} of the storage devices increases. However, the shape of the curve is more interesting. At low detection and repair times, there is a high positive impact on the MTTF of the system. However, as we increase the repair times, the system MTTF drops sharply. Interestingly, for a repair times greater than 120 days, about 1/9 of the MTTF of the storage device, the effect on the system MTTF of the detection/repair module is small. Thus, the detection and repair times must be much lower than the storage device failure MTTF to have a significant effect on the MTTF of the Archival System.

What is the optimal solution for a given MTTF with respect to disk MTTF repair times? The answer depends on the cost assigned to those two factors. By looking to all the curves of Figure 9, we can observe that we can achieve similar MTTF by using better media or by reducing the detection/repair times. For instance, a system that uses a storage device with ϕ_{sto} of 5 years (i.e., a low quality storage device) and has a detection/repair time of 30 days, is as good as a system that uses a high quality storage device with ϕ_{sto} of 20 years, but is only checked and repaired every 360 days. The decision of which alternative to choose will depend on the cost of the storage device versus the cost of more frequent detections.

We now expand our experiments by allowing formats to fail. In Figure 10 we show system MTTF as a function of ϕ_{sto} when formats can fail. We fix ρ_{sto} at 60 days, and now formats can fail with $\phi_{form} = 20$ years. To avoid introducing additional factors in the analysis, we assume site failures still cannot happen. Format failures are detected and repaired with $\rho_{form} = 60$ days. For comparison purposes, we have included in Figure 10, the results presented in Figure 8. The important conclusion that we can derive from the figure is that in an archival repository we cannot focus on single component types. It is surprising that even though the format MTTF is much larger than the disk's MTTF, the failure of formats still has a significant impact. This is because a document is lost if there is a disk failure *or* a format failure. The result is that we are taking the "worst" of those two failures, resulting in a system with a low MTTF. The result of this experiment shows that we need a comprehensive model, like the one proposed in this paper, to realize the interactions between the components and their effects on system MTTF.

Our model can be used to explore other possibilities that may improve reliability. For example, we now consider what happens if we are able to increase the number of copies maintained in the sites from two to three. In Figure 11, we maintain the same parameters at the same base values, but now each site has three copies in three different formats. In the figure, we can see that by increasing the number of copies to three, the MTTF increases from 34 years to 2101 when ϕ_{sto} is equal to 3 years. Although increasing the number of copies to three will undoubtedly increase the cost (as we need an additional 33% disk space and the need to handle an extra format), we have achieved an important improvement in the MTTF of the system.

As we stated earlier, a comprehensive model is important to get an accurate picture of the reliability of the system. In the next experiment, we use our system to explore a different, more complex failure distribution for storage devices. In this new distribution, we want to include the issue of "infant mortality."

When the failure distribution includes infant mortality, storage devices have a higher failure rate in the beginning of their life (in our case, 30 days) than in the rest of their lives. This can be expressed as a distribution that has a low MTTF within the first 30 days and a higher MTTF after that. The MTTF after the 30-day period will be 5 years. We will vary the

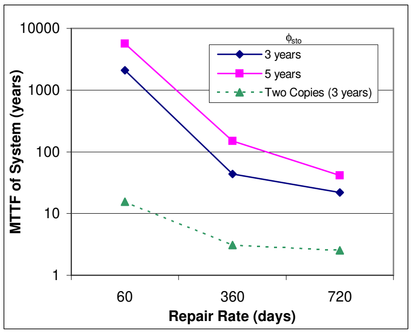

Figure 11: MTTF with 3 Copies (logarithmic scale)

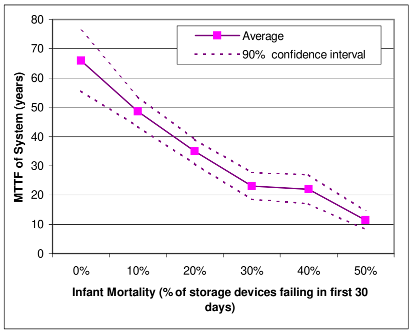

Figure 12: MTTF with Infant Mortality

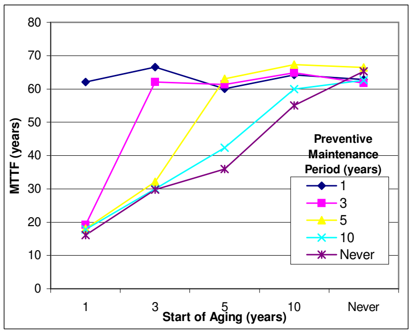

Figure 13: System MTTF with PM and Aging

early MTTF between 44 and 285 days. In this experiments we will assume that formats cannot fail. All other assumptions and repair procedures of the previous experiments are maintained (see Figure 7). In Figure 12 we show the system MTTF at given percentages of storage devices that fail in the first 30 days. For example, with an early MTTF of 285 days, 10% of the devices will fail within 30 days. With a MTTF of 135 days, 20% will fail. As expected, the higher the infant mortality, the lower the MTTF of the system. At a 0% infant mortality, the system MTTF was 65 years, dropping to 48 years when the infant mortality is 10% and dropping to only 11 years at 50% infant mortality level. Given this, when using components that suffer from infant mortality, a way to increase the MTTF of the system is for the failure detection module to check new components much more often than older components.

We now explore the issue of aging of storage devices. With aging, a storage devices will have a lower MTTF at the end of its life. For example, a disk may have a 5 year MTTF during its initial life and a MTTF of 2 years when it reaches its "aging" phase. In this experiment we will assume that formats cannot fail. All other assumptions and repair procedures of the previous experiments are maintained (see Figure 7). We evaluated the MTTF of the system for different points when aging starts (no graph shown). As expected, the sooner aging starts, the lower the MTTF of the system. If aging never occurs, the MTTF of the system is 65 years. If aging starts after 1 year, the system MTTF is 17 years, increasing to 42 years when aging starts after 5 years. Given this, when using components that suffer from aging, a way to increase the MTTF of the system is for the failure detection module to check old components much more often than newer components. Moreover, we should consider replacing old components with newer ones before the old components fail.

As a final experiment, we will evaluate the impact of Preventive Maintenance (PM) on a system with aging disks. Specifically, we will replace old disks with new ones before the old disks are expected to fail. This is done by copying (instantaneously) all documents from the old disk into a new disk, and then removing the old disk. In this experiment, we are assuming that disks do not have infant mortality and that disks have a 5 year MTTF during their initial life and a MTTF of 2 years when they reach their "aging" phase. Figure 13 shows five PM schedules for disks that age at different points. From the figure we can see that the most efficient PM schedule is one that matches the start of the aging period of the disk. For example, when we use a 10-year PM plan a system with disks that age after 5 years, will have a MTTF of 42 years. When we never perform PM, the system MTTF does not increase significantly. However, when we use a 5-year PM plan, the MTTF of the system increases to 63 years. If we keep increasing the frequency of the PM plan, the MTTF does not improve much more.

7 Conclusions

In this paper, we have studied the *archival problem*. We studied the different options for recovery and preventive maintenance, developing a comprehensive model for an AR. We described a powerful simulation tool, ArchSim, for evaluating ARs and the available archival strategies. We described how ArchSim can efficiently perform large simulations many components and very long durations. We demonstrated the use of ArchSim with a case study for a hypothetical Technical Report repository operated between Stanford and MIT. We considered options such as disks with different reliability, number of copies, format failure handling, and preventive maintenance. We believe ArchSim can help librarians and computer scientists make rational decisions about preservation, and help achieve better archival repositories.

8 REFERENCES

1. W. David Kelton Averill M. Law. *Simulation Modeling & Analysis*. McGraw-Hill, 1991.
2. John W.C. Van Bogart. *Magnetic Tape Storage and Handling*. National Media Lab, 1995.
3. C. Borgman, S. Chen, H. Garcia-Monlina, K. Thibodeau, , and G. Wiederhold. *NSF Workshop on Data Archival and Information Preservation*. National Science Foundation, March 1999. At http://cecssrv1.cecs.missouri.edu/NSFWorkshop/.
4. Arturo Crespo Brian Cooper and Hector Garcia-Molina. Implementing a reliable digital object archive, 1999. Submitted for publication to ACM DL 2000.

5. E. Çinlar. *Introduction to Stochastic Processes*. Prentice-Hall, 1975.

6. Yuan Chen, Jan Edler, Andrew Goldberg, Allan Gottlieb, Sumeet Sobti, and Peter Yianilos. A prototype implementation of archival intermemory. In *Proceedings of the Fourth ACM International Conference on Digital Libraries*, 1999.

7. Arturo Crespo and Hector Garcia-Molina. Archival storage for digital libraries. In *Proceedings of the Third ACM International Conference on Digital Libraries*, 1998. Accessible at http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1998-0082.

8. Arturo Crespo and Hector Garcia-Molina. Modeling archival repositories for digital libraries. Technical report, Stanford University. At http://www-db.stanford.edu/ crespo/papers/ArchSimFull.ps, 1999.

9. John Garrett and Donald Waters. Preserving digital information: Report of the Task Force on Archiving of Digital Information, May 1996. Accessible at http://www.rlg.org/ArchTF/.

10. Andrew Goldberg and Peter Yianilos. Towards an archival intermemory. In *Advances in Digital Libraries*, 1998.

11. G. Gordon. *The Application of GPSS V to discrete System Simulation*. Prentice-Hall, 1975.

12. Joseph Halpern and Carl Lagoze. The Computing Research Repository: Promoting the rapid dissemination and archiving of computer science research. In *Proceedings of the Fourth ACM International Conference on Digital Libraries*, August 1999.

13. Andreas Reuter Jim Gray. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publisher, 1993.

14. Kranch. Preserving electronic documents. In *ACM Digital Library Conference*, 1998.

15. T.I. Oren. Application of system theoretic concepts to the simulation of large scale adaptive systems. In *Proceedings of the 6th Hawaii International Conference on Systems Sciences*, pages 435–437, 1973.

16. Dhiraj K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall PTR, 1995.

17. Jeff Rothenberg. Avoiding technological quicksand: Finding a viable technical foundation for digital preservation. Technical report, Council on Library and Information Resources (CLIR), Washington DC, 1999.

Appendix I: MIT/Stanford CSTR Scenario Parameters

- AR Description
  - – Initial collection: num_{doc} documents. Each document, d, will have the following four materializations:
    - $\langle d, MIT, disk_i, form_j \rangle$,
    - $\langle d, MIT, disk_k, form_l \rangle$,
    - $\langle d, Stanford, disk_x, form_j \rangle$,
    - $\langle d, Stanford, disk_y, form_l \rangle$.

    Where MIT and $Stanford$ are the two sites; $disk_i$, $disk_k$, $disk_x$, and $disk_y$ are different storage devices; and, $form_j$ and $form_l$ are two different formats.
  - – Number of components and types: n_{sto} storage devices, n_{form} formats, 2 sites.
  - – Failure dependency graph: $site \rightarrow disk$, when the disk is in the given site.
- Distributions
  - – Disk Failure distribution during access: $U(1/\phi_{sto})$
  - – Format Failure distribution during access: $U(1/\phi_{form})$
  - – Site Failure distribution during access: $U(1/\phi_{site})$
  - – Disk Failure distribution during archival: $U(1/\phi_{sto})$
  - – Format Failure distribution during archival: $U(1/\phi_{form})$
  - – Site Failure distribution during archival: $U(1/\phi_{site})$
  - – Disk Failure Detection distribution: instantaneous
  - – Format Failure Detection distribution: instantaneous
  - – Site Failure Detection distribution: instantaneous
  - – Disk Repair distribution: instantaneous
  - – Format Repair distribution: instantaneous
  - – Site Repair distribution: instantaneous
  - – Document creation: num_{doc} documents at startup, then no documents are created.
  - – Document access rate: irrelevant as failure distribution during access is the same as during archival.
  - – Access duration rate: irrelevant as failure distribution during access is the same as during archival.
  - – Document selection: uniform over the num_{doc} documents.
- Policies
  - – Document Creation policy: for each document, four materializations are created, 2 in each site. In each site, each materialization is created in a different disk and in a different format.
  - – Document to Materialization: read from any materialization.
  - – Failure detection algorithm: complete scan of all disk, formats, and sites every τ_{sto}, τ_{form}, and τ_{site} days, respectively.
  - – Damage Repair algorithm: discard bad component and replace with new component taking δ_{sto}, δ_{form}, and δ_{site} days, for disks, formats, and sites respectively.
  - – Failure prevention algorithm: none
- ArchSim Parameters
  - – Stop Condition: when losing the first document.
  - – Simulation time unit: days.