Simulating the Effect of Preservation Actions on Repository Evolution

Christian Weihs, Andreas Rauber
Vienna University of Technology
Vienna, Austria
{weihs,rauber}@ifs.tuwien.ac.at
http://www.ifs.tuwien.ac.at/dp

ABSTRACT

One of the most important challenges in planning and maintaining a digital repository is to predict the needed resources on a long term basis, especially storage size and processing power. The main problem emerges from the need to migrate the data at certain times to newer file types, which takes time and alters the needed storage space, potentially branching into several migration paths for individual objects. Understanding the effect of different policy decisions, such as when to migrate or whether to stay within a format family or branching into several format families turns into a complex task, specifically when considering non-trivial ingest structures and assumptions on format evaluations. In this paper we present *ReproSim*, a framework that simulates the evolution of a digital repository and helps predicting these factors. We demonstrate the complexity and power of simulation to assist in preservation decisions in a set of scenarios involving different ingest and preservation planning profiles.

1. INTRODUCTION

Designing and operating a digital repository is a complex task. Especially estimating the scaling of the repository system, i.e. estimating the required storage space and computational power, across time considering a range of environmental options poses non-trivial challenges. While assumptions about the number and expected size of new objects to be ingested can be made with some diligence, the need for preservation actions to keep digital objects accessible adds significantly to the complexity. Following a migration strategy as one feasable way to maintain objects accessible, objects are converted to (potentially several) new formats at certain intervals in time, where the frequency of such migrations usually will depend on the validity and accessibility of a specific format (family). Thus, after a certain number of iterations, each object may exist in several versions, branching into a tree of different format (families), each of which again will be subjected to subsequent migrations. Identifying the effect of certain migration policy decisions, i.e.

- when to migrate: at ingest? when a specific format version is due to loose support? two months after the next-plus-one generation of the format comes into being?
- in how many paths to migrate: just within the format family? convert into more stable alternative formats that require fewer subsequent migrations? combinations?
- which tool (complexity) to use for migration: e.g. computational requirements such as more resource-demanding better-quality tools vs. simpler tools for mass-migration, effects on storage efficiency of the resulting objects
- for which files to apply these strategies, depending on file size, ingest type,..

is a complex issue. Identifying when peaks in computational resources for mass migrations are to be expected, or how storage requirements will grow, and how these change as a consequence of more risk-averse or risk-taking preservation policies requires detailed simulation of a repositories behavior based on explicitly modelled assumptions and specifications. This allows to understand the effect of certain policy decisions, specifically with respect to the branching factor of migrations into several target formats, providing a better basis to understand the trade-off between less risk (several copies in different formats) vs. more focused strategies.

Given the complex dependencies of such format decisions, bundled with non-linear growth both of the number as well as the size of objects to be ingested results in repository configurations that make straighforward calculation of its evolution unfeasible. Simulation offers a powerful approach to better understand the characteristics of a repository as it evolves under certain assumptions. Specific scenarios can be modelled and compared against each other, the effect of different policies can be analyzed, with subsequent decisions being based on the result of clearly specified simulation parameters rather than mere estimates. These, in turn, allow a monitoring of the validity of the simulation, as the actual evolution of crucial parameters such as ingest volumes, format validity periods, as well as the computational and storage costs of specific actions are tracked. This provides solid guidance in managing complex repository systems dealing with large volumes of heterogeneous material that are to be preserved over time.

To facilitate this process we have developed a simulator, that can show how a certain repository configuration will look like after several decades under a range of conditions that can be specified flexibly as a set of simulation parameters. It supports the specification of a repository configuration (file types, sizes and ingest timestamps/frequency, as well as future evolution of these) based on configuration files, as well as based on an existing collection profiles. Different migration rules can be specified, and the effect of these subsequently verified when running the simulation. The state of the archive at each point in time in terms of computational resources, storage requirements, and the number of versions of each object as well as of entire subcollections can be evaluated from the resulting data structures and logs.

The remainder of this paper is structured as follows: Section 2 reviews related work on preservation planning and collection profiling, forming the basis of simulating a repository's evolution. Section 3 describes the architecture and simulation parameters for the repository simulator. Exemplary simulation runs are presented in Section 4, followed by a short summary and outlook on future work in Section 5.

2. RELATED WORK

While being commonly used to understand the behavior of complex systems, modeling and simulation do not have a strong history in the analysis of digital repository systems. One of the few systems simulating aspects of a repository is ArchSim [7]. Focusing on storage technology, it allows to simulate the mean time to failure of an archive based on a library of failure distributions. The probability of not being able to interpret a format as it ages and becomes obsolete, for example, is modeled by a Weibull distribution. Different failure models can be assumed for different storage technologies. Using a complex architecture of triggers allows efficient modelling of failure probabilities over long periods of time. ArchSim/C [8] explicitly models costs associated with operating archival storage, including costs for creating, operating, monitoring and repairing a complex storage system. In a related line of work, a modeling approach is presented in [6] to analyze the reliability of system configurations for digital preservation. Again, the focus is on understanding the effect of component failures within a storage system. While these studies focus on understanding system failure characteristics and associated costs, the simulator presented in this paper focuses on understanding the evolution of individual files across a series of migrations into multiple branches, and the associated requirements in terms of storage and computational resources.

Testing and evaluating the effect of preservation action has been more intensively addressed from a planning perspective [2, 4]. Specifying the requirements for a specific preservation challenge (also referred to as objectives) and measuring how well different tools perform on selected sample data provide solid evidence for decisions on which preservation action component to deploy, and that component's effect on the object (in terms of significant properties retained), the storage space required, as well the the complexity of the deployment with respect to system and human resources that need to be provided. Preservation planning thus, on the one hand, provides valuable input to the simulator, concerning information on the processing requirements of certain types of preservation action tools as well as the resulting changes in object storage size. These can be obtained either directly from measurements obtained in preservation planning [3] or dedicated benchmark experiments measuring tool performance and the effect of preservation actions in controlled settings [1]. On the other hand, the repository simulator presented in this paper provides valuable input to a preservation planning process by providing a basis to estimate the costs associated with a certain preservation action, thus effectively closing the loop between planning and evaluation in these criteria.

In addition to the model parameters specified for a simulation run, more realistic initial configuration can be obtained from collection profiling services, as well as format registries such as PRONOM [5], which provide consolidated information on the lifetime/support time for selected formats, as well as offering a basis for analyzing the evolution of formats.

3. SIMULATING REPOSITORY EVOLUTION

The goal of the repository simulator is to offer the possibility to specify the content and ingest behaviour of a repository, then simulate migration rules on the files in the repository based on preservation plans, and collect statistical information about the changes in the repository.

During the simulation basically three different types of events get processed:

- **Ingest of new objects:** New files are added to the repository. The characteristics of this input stream, specifically date, initial file type and object size can be configured.
- **Migration of a file:** A file needs to be migrated to one or several other file types. The moment a migration has to take place can be specified by a set of rules depending on a range of factors, for example the expiration date of the file type or the size of the file.
- **Collect statistics:** On a regular basis statistical information is collected and stored in a file for later evaluation. This includes average file size, the number of executed migrations and so on.

3.1 Architecture

The Repository Simulator is realized as a Java application. The simulated archive is stored in a MySQL database, accessed via Hibernate. The following objects are mapped into the database model:

- **StoredFile:** Each ingested object is stored as a file stub (i.e. a file's profile) in the database. If a file gets migrated, a new file instance's profile is stored to the archive.
- **FileType:** Describes the available file types. Every file type consists of a type family name (ie. "Word Document") and a subtype name, which specifies the exact type version (ie. "Word 6.0"), as well as the average date of validity and the periodicity with which new versions are released.

- **MigrationTool:** The main properties of a migration tool are the duration of the process and how the file size is altered during the migration.
- **MigrationRule:** A migration rule defines when and how a migration should take place. This includes conditions which need to be met for the rule to be triggered (e.g a rule should only trigger for files smaller/larger than a certain size), the scheduled moment of the migration, the destination type (or a list of types, if for example a word document should be transformed into a PDF and a plain text file) and the used tools.

The model provides all information on the processes in the repository. On each migration event the new file is linked with its direct ancestor and with the tool element of the migration path. Additionally a generation counter gives direct insight how often a certain file got migrated. Furthermore, each file carries the information when it has been generated, by which tool and which rule. This makes it easy to track down the complete history of a certain file and allows to analyze the number of versions present of each original file or group of files, the percentage of active (ie. leaf versions in the migration tree) or inactive (ie. files that have already been migrated to newer versions) as well as the storage space used by these. (Note that this distinction between active and inactive files is currently rather basic. More complex representations may need to be modeled to account for the fact that an object may be migrated to a different format family to mitigate potential risks by having two different active versions in two format families. Extensions such as these are currently being implemented as part of the first evaluation cycle of the system. Similarly, delete orperations are currently being added to account for specific delete operations on migration, e.g. always keeping the original but deleting intermediate versions.)

3.2 Configuration

The configuration of a repository is done in plain text *ini*-files. There are basically four types of configuration files needed to specify all aspects of the simulation. In the following the core configuration possibilities are listed. To make the configuration flexible, many fields in the configuration are parsed with an expression language library, which means that any mathematical term can be used. Those fields are: the quantity and file size in the ingest configuration, the term and condition fields of the migration rule, and resulting change in file size and the computational cost of the migration process for each tool, expressed either in computation time or e.g. processor cycles (both of which can subsequently be normalized across time as the computational power of the underlying hardware infrastructure evolves). Note that file sizes in the configuration have no special magnitude (bytes, kilobytes etc.) associated with them. They can be specified in any magnitude, as long as it is in all configuration files.

- **Repository:** In this file the start and the end of the simulation is configured, as well as (the sequence of) all ingest events. It basically describes the characteristics of the repository to be simulated.

  The base configuration can either be obtained by a collection profile from an existing archive, by providing a list of file profiles, or by specifying groups of objects and their ingest characteristics by listing the number of objects, the mean and standard deviation in file size. This will create the according set of objects following e.g. a Gaussian or Weibull distribution. Similarly, the timeline of the ingest process can be modelled, so not all files are ingested at the beginning of the simulation, but the repository can grow step by step via ingest of original objects (in addition to the migrated ones) during the simulation process. in the case of modelling an existing repository for simuation purposes, the state of the repository needs to be provided as a collection profile detailing either the individual objects and their characteristics (format, sizes, ingest timestamps), or as a more compressed representation creating a model of the repository. Furthermore, parameters allow the specification of the growth characteristics of an archive, both in terms of number of objects and the average file size. These parameters can either be estimated or taken from the current history of a repository, e.g. the increase in average filesize of a collection of digital photographs or powerpoint files across the years, as well as the increase in numbers. These can be specified via almost arbitrary complexity, ranging from simple linear growth to more complex functions fitting real-life growth curves. Additionally, in combination with the format family configuration described in more detail below, the ingested objects will be of a specific version of the given file type families according to the timstamp within the simulation progress. Starting the simulation then creates the respective "files" as simulated entities with the respective ingest timestamps in the database.

  Listing 1 shows a sample repository configuration. In this case the simulation runs over 40 years starting in 2010/01/01 and ending 2049/12/31. In the beginning 1000 files of the type "doc" are inserted. They have an average size of 10000 with a deviation of 500. The next files are added in the year 2020: 3000 jpg files with average size 25000, deviated by 700. This ingest group has also specified the attributes "successive interval" (with value 2) and "successive count" (value 10). This way it is possible to specify a repetitive ingest, in this case every 2 years the same ingest is repeated for a total of 10 times.

  To make the configuration easier there is a simple collection profiling tool included that takes the repository structure from an existing archive, allowing one to operate on a real-life object type distribution. This currently reads objects from a mounted files system and can be adapted to meet API requirements of specific repositories. Alternatively, a statistics report that may be exportable from a repository can be converted to match the configuration file. Currently, the evolution parameters have to be estimated from an existing collection profile manually, with plans to provide this as an integrated module being currently evaluated.

Listing 1: Repository Configuration

```
[repository]
 simulation_start=2011/01/01
```

```
simulation_end=2050/12/31

[ingest1]
 Type=doc
 quantity=1000
 filesize = Dist:normal(10000,500)
 ingest_date=2011/01/01

[ingest2]
 Type=jpg
 quantity=3000
 filesize = Dist:normal(25000,700)
 ingest_date=2020/01/01
 successive_intervall=2
 successive_count=10
```

- **Filetype:** This describes a file type family. Each file type family consists of a family name and several subtypes with a specified time frame of how long they are supported, as well as how frequently new subtypes are generated. This results in a set of available file types at each point during the simulation, with objects being ingested as new originals usually being created in the most recent version of a file format family available. More complex configurations of mixes during overlap periods of format version validity are in principle possible using a number of distribution functions such as Gaussians or Weibull distributions.

  Listing 2 provides the configuration for the format type family "video". A subtype "xvid01" is created which is valid from the year 2000 to 2011. To model a sequence of consecutive subtypes a repetition group with the last two properties in the example can be specified. In this case 50 subtypes are created, shifted by 5 years, so the second one is valid from 2005 to 2017. Note that these subtypes can be specified at different levels of granularity, either, as shown in the example below, simple in the form of "AVI" files, or at a more detailed level, representing a range of video codecs embedded in an AVI container as individual subformats. This allow a realistic recreating of repository settings. Again, in principle the specifications of format version validities need to be specified in the simulation model. These settings could, in principle, also be imported from format registries.

Listing 2: file type configuration

```
[FileType]
name=video
extension=.avi
type=video

[subtype1]
subtype=xvid01
created=2000
expired=2012
successive_intervall=5
successive_count=50
```

- **Migration rule:** The migration rule is the most important part of the configuration. Each rule has an effective date, a source type for which the rule should be triggered, and a condition (for example "current file size is smaller than 5000") that determines for which files the rule should be executed. This allows for a rather fine-grained specification of migration policies, e.g. migrating smaller objects to multiple formats, whereas very large files might be migrated only within a single format family strand. Furthermore, migrations can either be based always on the most recent format version of each object, or always be based on the originally ingested objec, i.e. the root object, by setting the "source" parameter of the migration rule.

  Beyond that one or more destination types along with the tools to be used for the simulated migration can be listed. For each destination type one can again specify a condition, so complex migration policies can be mapped into the simulation model.

  Listing 3 provides a rule to migrate all files of the type family "doc" to the format families "pdf" and/or "rtf". The property "term" describes when the rule has to trigger, in this case two months before the respective sub-version of the file format expires. Two destination types are specified. The first one is a type of the family "pdf". The subtype is not specified directly, but with the keyword "maximal step" it is indicated that we want to migrate to a type from that format family which is available at migration time and has the longest expiration time. The destination subtype for rtf is specified as "minimal step", which means that the subtype with the next higher expiration date should be taken.

  Both destinations have a condition specified. The migration to each destination is only executed if it evaluates to true. In this example files smaller than 15000 are migrated only to rtf and files larger than 10000 only to pdf. Files with a size between 10000 and 15000 are migrated to both destination formats. (This example is only supposed to demonstrate the flexibility of configurations, allowing to address space considerations that may appear in real preservation planning scenarios, when certain objects types that may be in demand by different user communities should be made available in different formats, whereas other, potentially very large files, should not be kept in duplicate versions. It is not supposed to represent a recommended preservation plan within the scope of this paper. The same applies to the timing settings provided, i.e. whether a migration should happen 2 months prior to the expiry date.)

Listing 3: migration rule configuration

```
[migrationrule]
description=migration for doc files
term=subtype_expired -2*month
source_Type=doc

[destinationformat1]
destination_Type=pdf
destination_SubType=[maximal_step]
condition=file_size > 10000
tool=doc2pdf
source=current
```

```
[destinationformat2]
destination_Type=rtf
destination_SubType=[minimal_step]
condition=file_size < 15000
tool=doc2rtf
source=root
```

- **Migration tool:** For each virtual migration tool one can specify how the size of the file is changed during the migration process and how long the migration will take. Both the file size and migration duration are specified using mathematical expressions. Thus, one is not bound to simple linear changes but more complex effects can be simulated. Both units are dimensionless, i.e. as for th especification of the file sizes, these can be given in Bytes, Kilobytes, etc. More importantly, for the simulation of computational resources, either computation time or e.g. processor cycles may be specified. The latter may proove useful when more realistic estimates of the computational requirements are required. By mapping operation cycles in a virtual unit, normalization factors may be applied to account for improvements in processing power over time. Still, in first experiments, specifying actual processing time, and then applying a normalization factor to account for improved computational facilities, seemed to be more easily accepted. Note, that the primary use of the effort simulation is not a precise determination of the HW requirements at a specific point in time in the future, but to capture the potential of cumulative effects resulting from certain preservation policies. These may stem, for example, from the difference of migrating on-ingest (usually leading to a more even spread of subsequent migrations) or on-expiry - resulting in strong peaks if all objects of a specific format version need to be migrated to, e.g. the subsequent version.

  Two examples:

  - **size=currentsize * Math:log(currentsize)**: This specifies a logarithmic growth of the file. (The keyword "Math" in this string references the Java class java.lang.Math, which has methods for many mathematical operations, all accessible through this keyword.)
  - **duration=Math:max(currentsize * 3, 18000)**: The migration should take three times as long as the file is big, but at least 18000ms.

From this set of configurations the simulation of a repository's evolution is started. For each (set of) files specified in the repository configuration, the according sets of files are "created" as database entries with the respective timestamps. For each of these the respective migrations based on the preservation planning triggers as specified in the migration rules setting are executed consecutively. Thus, for each file specification in the database meeting a migration condition the respective new file(s) are generated with new ingest timestamps and file sizes considering the migration time needed and the file size change incurred as specified in the respective migration tool specification. From these, the resulting hypothetical computational load (i.e. the number of files to be migrated at any specific point in time) and the required storage space for the accumulated archive can be calculated. (Currently, the simulator only supports single-processor migration, i.e. all pending migrations are executed consecutively. An extension allowing the specification of a (growing) multiprocessor architecture or simulated cloud support to scale with the repository is under investigation).

4. EVALUATION

For evaluation we performed a series of simulations to see whether the simulation works correctly, and to what extent the available configurations are flexible enough to support realistic scenarios. In the following one of our sample configurations is presented along with the results of the simulation.

4.1 Configuring a repository

In this simulation, we defined 6 format families. Note that the types may not conform to reality, because they are just an assumption to provide a simple simulation example - in principle, these could also be modeled as entirely abstract format types that have certain characteristics such as stability, support, etc. that are relevant for the aspects covered by the simulator. The same applies to the migration tools, which can be specified in an equally abstract way. We decided to choose real file types for the sake of clarity and ease of discussion. Please note, again, that the validity periods specified do not correspond to real values, which would need to be obtained from format registries or from an analysis of the evolution of object formats in an existing repository. The same applies to the file sizes. The types are the following:

- **jpg:** for compressed pictures; every subtype is valid for 28 years and every 15 years a new subtype becomes available.
- **tiff:** for uncompressed pictures; valid for 44 years with a new subtype every 30 years.
- **wordDoc:** for proprietary text documents; valid for 15 years, new subtype every 5 years.
- **openDoc:** for editable text in a free format; valid for 8 years and every 4 years a new subtype.
- **pdf:** valid for 16 years and a new subtype every 8 years.
- **rtf:** valid for 12 years and a new subtype every 6 years.

The simulation runs for 100 years and every year there are 100 wordDoc files with average size of 500 and 100 jpg files with average size of 1600 are added to the repository.

Several rules specify the migrations:

- **jpg:** The compressed pictures are migrated to the tiff format. The file size multiplies by 10 during the migration.

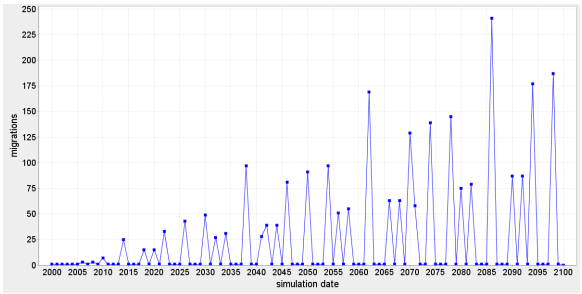


Figure 1: Number of migrations to be expected at each time interval

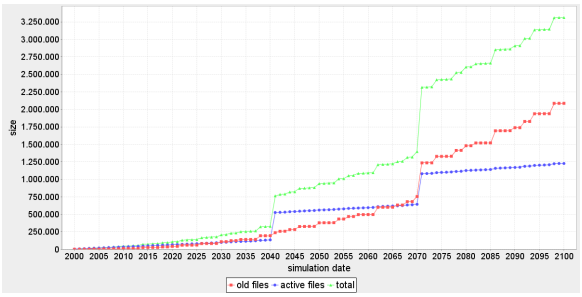


Figure 2: Number of files in the repository, subdivided in active files as well as earlier interim migration copies

- **wordDoc:** Word Documents get migrated both to openDoc, rtf and pdf. The file size grows 20 percent in case of openDoc and decreases by 5 percent to rtf and 10 percent to pdf.
- **pdf, tiff, rtf and openDoc:** These types get migrated to a newer version in the same family several months before they expire. All are migrated with maximal step size and the file size varies several percent downwards in case of tiff, upwards for the other ones.

For all migrations tools are created with the size changes as described above and named following the pattern "<source>2<destination>" (eg. "jpg2tiff").

4.2 Simulationg migrations

Figure 1 shows the number of migrations performed as the archive grows and format versions trigger migration. There are very few migrations in the first years of the simulation, but the number increases as time passes and more format versions expire. There are several peaks that show the moments multiple types expire at the same time. These peaks are a good hint that the preservation plan may be revised to avoid them or to plan for appropriate resources for mass migration projects at regular intervals. It also allows a more detailed evaluation of slight modifications of certain rules, e.g. starting migrations at earlier points in time, migrating always to the most recent version, potentially suffering from lower-quality tools available vs. migrating to a more stable version that already exists for a longer period of time, and others.

Figure 2 shows the storage space needed by the repository. The total size is growing constantly, with several boosts correlating partially to the peaks in Figure 1. Note that the two very big steps in the years 41 and 71 have no special spike in Figure 1. These two steps are the result of the expiring of a sub-version of the tiff format. We do not have especially many tiff files in the repository, but the files are significantly larger than the other files, so the migration of those files has a drastical impact on thel archive size assuming that the original files are not deleted.

At the same time, the proportion between the active files (representing the most recently migrated version of each file in each migration branch) and the old files (the copies left behind after migrating a file to a newer subtype) changes drastically. This provides a good impression whether a cleanup

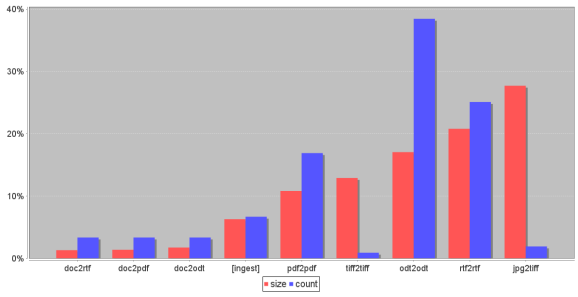


Figure 3: Tool usage, depicted as data volume handled by a tool and number of times it was called

strategy for the old data may be necessary to save on storage costs, potentially adapting preservation policies at an institution. (Different delete policies for inactive file formats, such as deleting the last-but-one, every second version, or keeping only the original and the most recent version, etc. are currently being evaluated as an additional configuration setting.)

In Figure 3 the usage of the tools is analyzed. The first bar for each tool family shows the percentage of the aggregated size of all files generated with this tool. The second bar shows how often this tool-family is called. In the diagram it is easy to see that the tools "odt2odt", "jpg2pdf" and "rtf2rtf" have the biggest potential to save space as they are responsible for 71% of the whole archive content. But one should keep in mind that "odt2odt" and "rtf2rtf" are called many times. In comparison "jpg2tiff" is called infrequently, so it might be ok to exchange it with a tool that is slower but has a smaller output.

Beside the statistics generated by the simulator, it is also of interest whether the simulation process is finished in reasonable time. For this simulation 259400 migrations were executed and the total process needed less than 15 minutes on a simple workstation, showing the feasibility to run sufficiently complex simulations.

4.3 Comparing policies

In the following example we assume an archive ingesting files from two format families. For ease of discussion, let's call them documents and images. We further assume that two different image subformats exist (e.g. jpeg and tiff), one

with a rather rapid release cycle of 3 years, the other with a slower cycle of 7 years, whereas for the document file formats we assume a single format family with a replacement cycle of 5 years. The individual format versions receive between 13 to 25 years of support, defining at each point in time a number of potential migration versions. To keep the graphs simple, ingests are kept growing linearly for all formats, with annual ingests. For preservation, jpegs are migrated both to jpeg and tiff, whereas both tiff and the documents are migrated within their respective formats. The two different policy strategies, and thus the only parameters varied in this simulation, concern the step-width for the migrations, i.e. whether we prefer to migrate each object to its next available version (min-step) or to the newest version available at the time of migration (max-step).

The results of this scenario are shown in Figure 4. The Min-step scenario obviously requires many more migrations, as multiple versions of each file are generated at rather short cycles whenever a format version expires, with the subsequent version expiring soon after. This results in increasingly high peak loads, especially in years when two format versions happen to expire at the same time. It also leads to a much higher number of "old" files, i.e. interim migration versions that soon surpass the number of objects actively used (note the different scales in the two graphs), calling for the evaluation of suitable deletion strategies, which are currently being added to the simulator.

5. CONCLUSIONS

Planning and operating a repository tasked with preserving heterogeneous sets of objects over long periods of time poses severe challenges when it comes to estimating growth in storage space as well as processing power required to run the required preservation actions. Also, from a preservation planning perspective, the effects of certain policy decisions such as when to migrate and how many copies to retain, are difficult to certain due to the complex behaviour emerging from multiply branching migration paths. Two different policies are devised, both relying on migration at ingest. In one case, objects are only migrated within the tiff family using the maxstep, i.e. migrating to the most recent format version available.

In this paper we presented *ReproSim*, a tool to simulate repository evolution over time. the strength of the approach lies in the flexibility offered by the configuration of the system, both in terms of modelling the content as received from the producers over time, as well as the content produced as a result of preservation actions, specifically migrations. Based on explicitly modelled assumptions of format stability, changes in object size induced by migrations, and others that can be explicitly specified (or, preferably, should be modeled based on an analysis of the history to date for the respective format types), different scenarios can be evaluated and compared, and the effect of different policies can be demonstrated. This, in turn, provides a better basis for policy, design, and structural planning decisions, helping both in the set-up and operation of a repository, as well as supporting preservation planning to evaluate the effect of certain recommendations.

While the simulator offers a very flexible basis for configuring different institutional settings, scenarios for object format evolution as well as preservation plans, a range of additional parameter settings emerged as being desirable from first case studies. Some of these, addressed already in the paper, include the possibility to simulate different object deletion policies, support for simulating multi-processor/cloud environments when evaluating peak loads, and others. A tighter integration with existing format registries, as well as more sophisticated collection profiling will allow better estimates for some of the core parameters in the system, specifically file format evolution and stability, as well as the characteristics of the ingest stream over time from different consumers. This will also allow verification of the simulation against existing repositories and their evolution to verify parameter settings and projections on file size growth. A tighter integration with preservation planning frameworks may help to provide closed feedback loops as well as offer better estimates on tool behavior with respect to resulting object sizes and processing times. Last, but not least, an improved interface helps with specifying the different scenarios and visualizing results in an integrated application.

6. REFERENCES

[1] Brian Aitken, Petra Helwig, Andrew Jackson, Andrew Lindley, Eleonora Nicchiarelli, and Seamus Ross. The planets testbed: Science for digital preservation. *Code4Lib*, (3), June 2008.

[2] Christoph Becker, Hannes Kulovits, Mark Guttenbrunner, Stephan Strodl, Andreas Rauber, and Hans Hofman. Systematic planning for digital preservation: Evaluating potential strategies and building preservation plans. *Intl Journal on Digital Libraries (IJDL)*, 10(4):133–157, Dec 2009.

[3] Christoph Becker, Hannes Kulovits, Michael Kraxner, Riccardo Gottardi, Andreas Rauber, and Randolph Welte. Adding quality-awareness to evaluate migration web-services and remote emulation for digital preservation. In *Proceedings of the 13th European Conference on Digital Libraries (ECDL 2009)*, volume 5714 of *LNCS*, pages 39–50. Springer, September 2009.

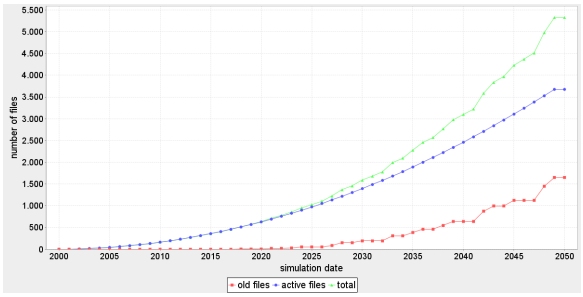
[4] Christoph Becker and Andreas Rauber. Decision criteria in digital preservation: What to measure and how. *Journal of the American Society for Information Science and Technology (JASIST)*, 62:1009–1028, 2011.

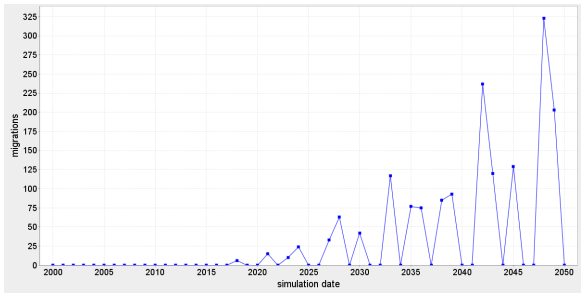
[5] Tim Brody, Leslie Carr, Jessie Hey, Adrian Brown, and Steve Hitchcock. PRONOM-ROAR: Adding format profiles to a repository registry to inform preservation services. *International Journal of Digital Curation*, 2(2), November 2007.

[6] Panos Constantopoulos, Martin Doerr, and Meropi Petraki. Reliability modelling for long term digital preservation. In *Proceedings of the 9th DELOS Network of Excellence Thematic Workshop on Digital Repositories: Interoperability and Common Services*, Heraklion, Greece, May 11-13 2005.

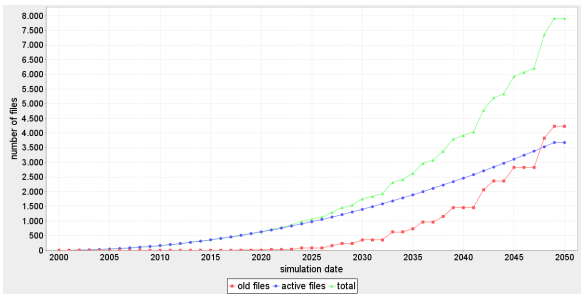
[7] Arturo Crespo. *Archival repositories for digital libraries*. PhD thesis, Stanford University, March 2003.

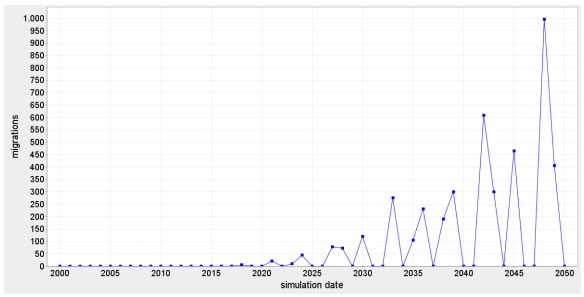
[8] Arturo Crespo and Hector Garcia-Molina. Cost-driven design for archival repositories. In *Proceedings of the First ACM/IEEE Joint Conference on Digital Libraries (JCDL'01*, pages 363–372, Roanoke, Virginia, USA, 2001. ACM Press.



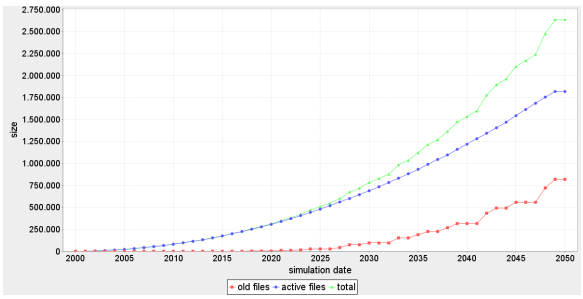


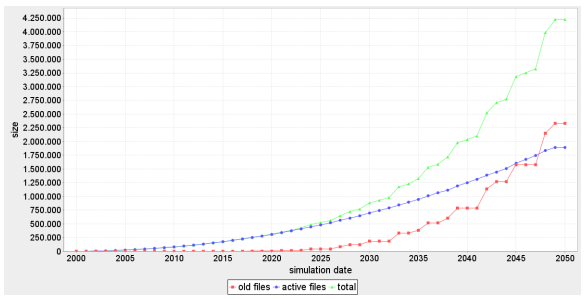
MaxStep: migrating to the newest available format version: (a) file count, (b) migrations





MinStep: migrating to the next available format version: (c) file count, (d) migrations





Resulting storage requirements: (c) MaxStep, (d) MinStep

Figure 4: Comparing migration step withs: Figs a and b show the number of files and migratons when migrating to the newest available file format version; Figs c and d show these for migrations to the respective next available versions. The resulting storage requirements and distributions between active and interim files is given in Figs (e) for MaxStep and (f) MinStep.