Shelter from the Storm: Building a Safe Archive in a Hostile World

Jon MacLaren, Gabrielle Allen, Chirag Dekate, Dayong Huang, Andrei Hutanu, and Chongjie Zhang

Centre for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803
{maclaren, gallen, cdekate, dayong, ahutanu, czhang}@cct.lsu.edu

Abstract. The storing of data and configuration files related to scientific experiments is vital if those experiments are to remain reproducible, or if the data is to be shared easily. The prescence of historical (observed) data is also important in order to assist in model evaluation and development. This paper describes the design and implementation process for a data archive, which was required for a coastal modelling project.

The construction of the archive is described in detail, from its design through to deployment and testing. As we will show, the archive has been designed to tolerate failures in its communications with external services, and also to ensure that no information is lost if the archive itself fails, i.e. upon restarting, the archive will still be in exactly the same state.

1 Introduction

The Southeastern Coastal Ocean Observing and Prediction (SCOOP) Program's Coastal Model Project [15], is an ongoing collaboration between the modeling research community and operational agencies, such as the National Oceanic and Atmospheric Administration (NOAA). The project aims to take today's cutting-edge activities from the research community, and develop these so they can form the basis for tomorrow's operational systems.

Part of this project's work involves the regular execution of coastal modeling codes, such as ADCIRC [13] and SWAN [11], for various geographical regions. Additional runs of some codes are performed to predict the path and effects of ongoing tropical storms and hurricanes; the results from these runs are passed to groups involved in evacuation planning. The project also tries to verify the accuracy of the coastal models by verifying the predictions the codes make against real-world observed data.

To support this work, a data archive was required which would store:

– atmospheric model outputs (wind data),
– results generated by the hydrodynamic models, which use the atmospheric model outputs for input (wage/surge data), and
– observational data to be used for verification of model results (sensor data).

© Springer-Verlag Berlin Heidelberg 2005

The archive would therefore form a backbone for the research efforts of the project, and as such, have to be both highly available, and reliable.

To meet this need, a data archive was constructed at the Center for Computation and Technology. Although supporting the SCOOP Coastal Model Project was our prime objective, we wanted to be able to re-use most of the archive's functionality, and code, for other efforts with data storage requirements, e.g. our group's numerical relativity work.

This paper describes the construction of this archive in detail. Section 2 briefly discusses data storage requirements, then describes the design of the archive service; Section 3 describes the archive's implementation. Section 4 describes how APIs and tools were developed to allow easy access to the archive, and Section 5 explains how good engineering practices were used to ensure reliability and code re-use. Finally, Section 6 explains some ideas for future work, and Section 7 gives our conclusions.

2 Design and Architecture

The data storage requirements for the SCOOP Project are simple. The files to be stored are small (no more than a few MB at most) so they can be easily moved to the archive.[1] In addition, there was no requirement to provide any kind of access control.

To complement the archive, a Metadata Catalog would be provided, which would store information about the model configurations used to generate the stored data. This catalog should be the first port of call for people looking for data outputs from the project, and can provide references to the location of the data, in response to users' searches. As the catalog is not the subject of this paper, it is not described here, although interactions between the archive and the catalog are.

The architecture for the archive is shown in Figure 1. In more detail, the steps for uploading files to the archive are as follows:

U1. The client contacts the archive, providing a list of files which they wish to upload to the archive. The archive decides where each file will be located within the archive.[2] The archive's response groups the original files into one or more **transactions**, each of which is associated with: a location in the storage area (specified by a set of URLs for various scheme names); a subset of the list of files; and an identifier which the client uses in later interactions.

[1] At the time we began construction of the archive, it was not clear to us what volume of data would be generated by the project each day, nor was it clear how long data needed to be kept for. We have since discovered that the project generates approximately 1 TB of data per month.

[2] Within the SCOOP Project, there is a File Naming Convention, allowing the archive to deduce metadata from the names of the files, and thus determine the location of each file within the archive's directory structure. Files belonging to the output of a single run of a model code will be stored in the same directory. New directories for new code runs are created automatically.

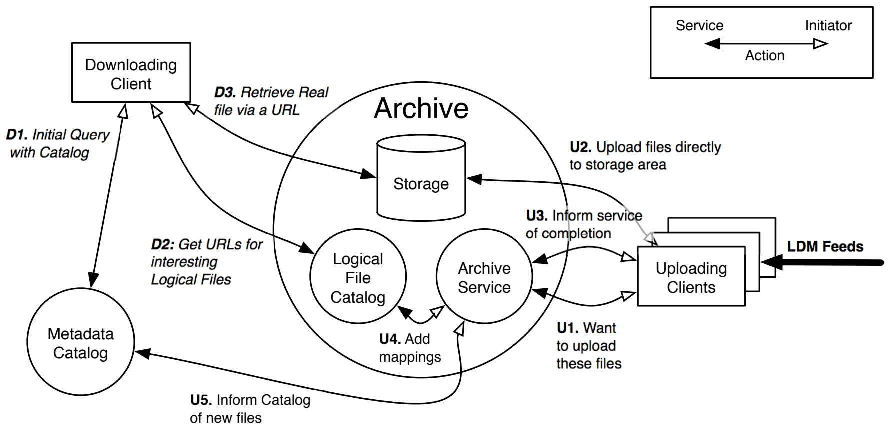

Service
Initiator
Action
Downloading Client
D1. Initial Query with Catalog
D2: Get URLs for interesting Logical Files
D3. Retrieve Real file via a URL
Archive
Storage
Logical File Catalog
Archive Service
Metadata Catalog
U1. Want to upload these files
U2. Upload files directly to storage area
U3. Inform service of completion
U4. Add mappings
U5. Inform Catalog of new files
Uploading Clients
LDM Feeds

Fig. 1. Basic architecture for the SCOOP Archive, showing steps for **U**ploading and **D**ownloading files

The following steps are then carried out for each of the transactions.

U2. The client uploads the files to the archive storage with some third-party software, e.g. GridFTP [8,2], and a URL given to it by the archive.

U3: request. After the client has uploaded the files to the location, it informs the archive that the upload is complete (or aborted), using the identifier.

U4. The archive contructs Logical File Names (LFNs) for the files which have been uploaded, and adds mappings to Logical File Catalog that link the LFNs to various URLs that can be used to access the physical files.

U5. The archive informs the catalog that there are new files available in the archive, providing the LFNs.

U3: response. The archive returns the LFNs to the client.

The steps for downloading a file from the archive are as follows:

D1. A client queries the **Metadata Catalog** to discover interesting data, e.g. ADCIRC Model output for the Gulf of Mexico region, during June 2005. Some Logical File Names (LFNs) are returned as part of the output, together with a pointer to the Archive's Logical File Catalog.

D2. The client chooses some LFNs that they are interested in, and contacts the Logical File Catalog service to obtain the files' URLs.

D3. The client picks URLs (based on scheme name, e.g. `gsiftp` for GridFTP) and downloads the files directly from the Archive Storage.

Note that this simply describes the sequence of interactions that are exchanged in order to achieve these tasks. We will later show that the clients indicated in the diagram can be either command-line tools, or a portal web-page.

3 Implementing the Archive Service

The architecture was refined into a message sequence diagram, and implemented using Web Services. Early on, we chose to use the Grid Application Toolkit (GAT) [4], developed as part of the GridLab project [3], to help with file movement, and also to provide access to Logical File services. The GAT wraps different underlying technologies (using "adaptors"), and so provides consistent interfaces. Here, the underlying Logical File Catalog is a Globus RLS, but this could be replaced with a similar component, without changing the archive's code. Our desire to use the GAT led us to select C++ as the language to use for implementing the service, which in turn led to us using the Web Service tooling provided by gSOAP [6,7]. Following the advice from the WS-I Basic Profile 1.1 [5, Sec. 4.7.4], we avoided using RPC/Encoded style[3] for our services. Instead we chose Document/Literal style, first designing the XML messages that would be exchanged, then producing XML Schema and WSDL definitions. From these definitions, the gSOAP tooling was used to automatically generate code stubs.

During the upload process, the archive passes back URLs for a staging area, rather than allowing clients to write directly to the Archive Storage. This also makes it simpler to prevent additional (i.e. unauthorized) files from being inserted into the archive. A distinct staging directory is created for each transaction identified by the archive.

4 Archive Interfaces and Tools

4.1 Downloading

We have provided two complementary mechanisms for clients to download data, namely:

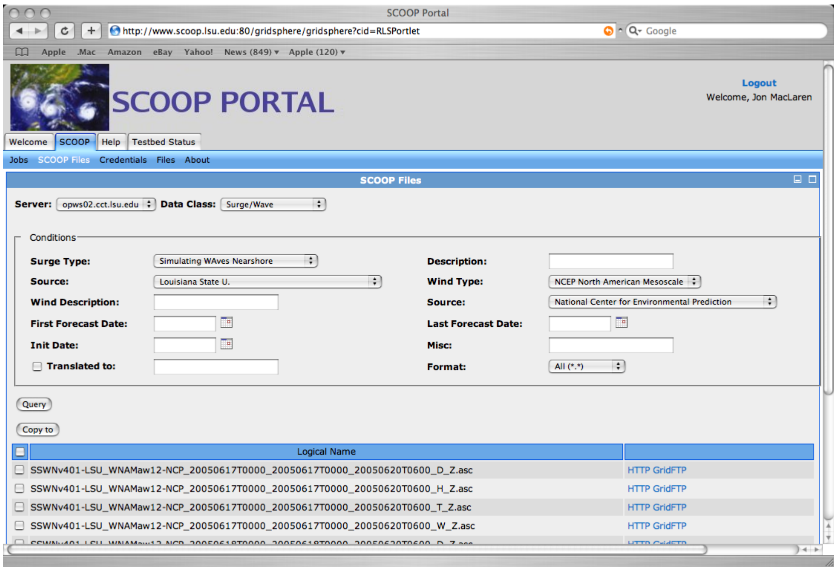
– Command-line tools, e.g. `getdata`; and
– A portal interface, built using GridSphere [9], an open-source portal framework,[4], also an output from the GridLab project [3], which uses JSR 168 compliant portlets [1].

The `getdata` tool has a simple syntax, encapsulating the client side of the message exchanges with the Logical File Service and the download from the Archive Storage, and can choose between different protocols for downloading the data. This was achieved using the GAT, making `getdata` easily extensible if new protocols need to be added. Currently, GridFTP and https downloads are supported.

Through the portal interface, users can access the same functionality as with the command-line tools. Users can search for files, and download them, either

[3] **Restriction R2706** (which also appeared in Basic Profile 1.0) states: "A `wsdl:binding` in a DESCRIPTION MUST use the value of "literal" for the `use` attribute in all `soapbind:body`, `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` elements."

[4] Available for download from `http://www.gridsphere.org/` at time of writing.


SCOOP Portal
http://www.scoop.lsu.edu:80/gridsphere/gridsphere?cid=RLSPortlet
Google
Apple .Mac Amazon eBay Yahoo! News (849) Apple (120)
SCOOP PORTAL
Logout
Welcome, Jon MacLaren
Welcome SCOOP Help Testbed Status
Jobs SCOOP Files Credentials Files About
SCOOP Files
Query
Copy to

Server: opws02.cct.lsu.edu Data Class: Surge/Wave

Conditions

Surge Type:	Simulating WAves Nearshore	Description:	
Source:	Louisiana State U.	Wind Type:	NCEP North American Mesoscale
Wind Description:		Source:	National Center for Environmental Prediction
First Forecast Date:		Last Forecast Date:	
Init Date:		Misc:	
Translated to:		Format:	All (*.*)

Logical Name	
SSWNv401-LSU_WNAMaw12-NCP_20050617T0000_20050617T0000_20050620T0600_D_Z.asc	HTTP GridFTP
SSWNv401-LSU_WNAMaw12-NCP_20050617T0000_20050617T0000_20050620T0600_H_Z.asc	HTTP GridFTP
SSWNv401-LSU_WNAMaw12-NCP_20050617T0000_20050617T0000_20050620T0600_T_Z.asc	HTTP GridFTP
SSWNv401-LSU_WNAMaw12-NCP_20050617T0000_20050617T0000_20050620T0600_W_Z.asc	HTTP GridFTP

Fig. 2. Screen capture from the SCOOP Portal, showing a typical search

through the browser, or perform a third-party file transfer via GridFTP. The portal interface, shown in Figure 2, integrates this and other capabilities, such as Grid monitoring, Job Submission and Visualization into a single interface.

4.2 Uploading

In order to simplify the construction of clients, a two-layered C++ client API was written and, like the service, was based on gSOAP. The first level of the API neatly encapsulates each of the two message exchanges, labeled **U1** and **U3** in Figure 1, into two calls `start_upload` and `end_upload`.

A higher-level API with a single call, `upload`, is also provided. This encapsulates the entire process, including the uploading of the files themselves. The `upload` call has the following signature:

```
bool upload(std::vector<std::string>& uploadFiles,
            std::string                urlType,
            bool                       verbose);
```

The provision of such a layered API makes the construction of tools far simpler.

Currently, files can only be uploaded via a command-line tool, `upload`, which allows a variety of transport mechanisms to be used to copy files into the archive. Even though the interactions with the archive service are more complicated than for downloading files, the command-line syntax remains trivial.

```
upload -gsiftp -done rm SSWN*.asc
```

This example will upload all ".asc" files in the current directory starting "SSWN" (produced using the SWAN code), transferring the files with GridFTP (GridFTP uses URLs with the scheme name "gsiftp") and will remove the files if they are successfully uploaded (specified by "-done rm").

5 Ensuring Stable, Robust, Re-usable Code

A key challenge when building distributed systems is tolerating problems with connectivity to external services. However, it would also be reckless to assume that our archive service would be perfectly reliable. No feasible amount of testing is sufficient to remove all the bugs from even a moderately sized program. In addition to the archive code, we are also relying upon the GAT and gSoap, (not to mention the Linux C library, compilers and operating system).

We have employed a number of techniques while designing the archive to make it as reliable as possible. These techniques, plus other "good practices" that we have employed, are described below.

5.1 Tolerating Failure in Remote Services

In the architecture shown in Figure 1, the Metadata Catalog is clearly a distinct component. However, in our implementation, from the perspective of the Archive Service component of the Archive, both the Metadata Catalog and the Logical File Catalog are remote components; only the Storage component is physically co-located with the Archive Service.

As stated earlier, remote components may become unavailable temporarily, only to return later. These **partial failures** are not encountered in "local" computing. If you lose contact with a program running on your own system, it is clear that some sort of catastrophic failure has occured; this is not the case in a distributed system. For an excellent discussion on partial failures and other inherent differences between local and distributed computing, the reader is directed to [12].

Here, we have tried to insulate ourselves from partial failure as far as possible. Following the advice in [12], we have made the partial failure explicit in the archive service interface. In the response part of interaction **U3**, when the user is returned the set of logical files corresponding to the newly uploaded files, they are told which of these logical files have been successfully stored in the Logical File Catalog.[5] The user knows that they need take no further action, and that the logical files will be uploaded to the Logical File Catalog when it becomes available again.

5.2 Recovering from Failures in the Archive

Interactions with the Archive are not stateless. Transaction IDs are created, and associated with areas in the Archive Storage, and with files that are to be uploaded. These IDs are used in later interactions and must be remembered by the Archive if it is to function correctly.

[5] Similarly, they are informed of whether or not the files have been registered with the Metadata Catalog.

Given what was stated earlier about the reliability of our own programming, and our operating environment, we chose to place all such state into a database located on the machine with the Archive Service. The "pending" insertions for the Logical File Catalog and Metadata Catalog (described in the previous section) are also stored in this database. Thus, if the service terminates for some reason, and restarted, it is able to pick up from exactly where it left off.

Note that we can also correctly deal with partial failure in the case where a transaction might be completed, but the response fail to reach the client. The client can safely retry the complete/abort transaction operation until they receive a response. If they receive a message stating that the complete/abort has succeeded, then they have just now terminated the transaction. If they receive a message stating that the transaction is unknown, then a previous attempt to complete/abort the transaction must have succeeded.

5.3 Keeping Domain-Specific Code Separate

Although the archive was primarily created for use in the SCOOP project, we have tried to keep project-specific functions separate from generic functions. Specifically, SCOOP uses a strict file naming convention, from which some of the file's metadata may be extracted. The filename therefore dictates where the file should be stored, etc. To keep the project-specific code separate, methods on a `FilingLogic` object are used to decide where to place all incoming files. Different subclasses of the `FilingLogic` class can be implemented for different "flavours" of archives.[6]

5.4 Summary

Through extensive testing, we have determined that the archive is stable. During initial trials, we used multiple clients to simultaneously upload files in rapid succession. Over one weekend, 20,000 files were successfully uploaded. The archive remained operational for a further three weeks (inserting a further 10,000 files), until a change to the code necessitated that it be manually shutdown.

During this time, we monitored the size of the Archive Service process. It seems that the program does leak a small amount of memory. After a number of months, this would likely cause the process to fall over. To prevent this happening, we have chosen to manually shut the service down every 14 days, and then restart. This "preventative maintenance" ensures that the archive does not fail unexpectedly.[7]

Although we have strived to make the archive as reliable as possible, there is a limit to how much we can improve the availability of the archive while it still resides on a single machine. The hardware in the archive machine is not perfect,

[6] Undoubtedly when the archive is first applied to a new project, there will be new requirements, and the FilingLogic interface will change. Nonetheless, this transition will be greatly simplified by the existence of this boundary.

[7] If for some reason, the archive needed to remain operational during the scheduled maintenance time, this could easily be moved or canceled (provided many successive shutdowns are not canceled).

nor are we using an Uninterruptable Power Supply (UPS). The campus network also causes periodic failures.

It seems that replicating the data archive would yield the biggest improvements in reliability.

6 Future Work

This first version of the archive provides us with a useful basis for future development. There are a number of ways in which we want to extend the basic functionality described above, the two most important of which are explained below.

6.1 Transforming Data on Upload/Download

Currently, the archive stores data in the form in which it is uploaded; downloading clients receive the data in this same format. We wish to support the following scenarios:

- The compression of large ASCII-based files when they enter the archive, and their decompression when they are downloaded (preferably after they have reached the client).
- The partial retrieval of a dataset. Some of the data stored in the archive is in NetCDF format [14], which supports retrieval of subsets of variables, ranges of timesteps, etc.
- Retrieval of data in different formats, e.g. retrieving single precision data from a double precision file.

To support this type of operation, we are proposing to associate a **specification** with each file that specifies the current format which the file is in, the type of compression, etc. Specifications are used at upload and download time; files may be transformed by the archive upon arrival.

6.2 Notification

One of the key goals of the SCOOP Project is to improve responsiveness to storm events, such as hurricanes, which are relatively common in the Southern United States. When a hurricane advisory arrives at the archive, it should trigger high-priority forecasts for the current location of the storm.

To support this work, we have recently implemented a simple interface that can be built upon to perform sophisticated patterns of notification. When a file is ingested into the archive, a message is sent to the `FilingLogic` object. The SCOOP implementation of this executes a script (forked to run in the background, so as to not affect the archive's performance), passing the Logical and Physical File Names as parameters.

6.3 Lifetime Management for Data

Currently, data is removed from the archive automatically after a fixed time. It should be possible for uploading clients to request storage for a particular duration. It should also be possible for this lifetime to be altered by other, authorized clients.

7 Conclusions

We have described the construction of a reliable data archive, constructed to satisfy storage requirements from a coastal modeling project. A number of techniques were employed, from the design phase through to the final testing, to ensure reliability.

We also showed how the archive was designed so that it could be re-used in other projects. In particular, we endeavoured to keep all project-specific code separate from the generic code, and provided an internal API which allows new project-specific code to be easily provided.

It is likely that future versions of the archive will rely on other systems for backend data storage. The most obvious candidate is the Storage Resource Broker (SRB) from SDSC [10], which provides excellent support for managing highly distributed data stores, and which would also satisfy some of our new requirements from Section 6, e.g. the retrieval of subsets of data.

Acknowledgments

This work was made possible by funding from the Office of Naval Research and the National Oceanic and Atmospheric Association, received through Louisiana State University's participation in the Southeastern Universities Research Association (SURA) Southeastern Coastal Ocean Observing and Prediction (SCOOP) program.

References

1. A. Abdelnur, S. Hepper, *Java^{TM}Portlet Specification Version 1.0*, Java Specification Request 168 (JSR 168), Commuinity Development of Java Technology Specifications, Oct. 2003. Online at: `http://jcp.org/en/jsr/detail?id=168`
2. W. Allcock (Ed.), *GridFTP: Protocol Extensions to FTP for the Grid*, Global Grid Forum Recommendation Document GFD.20. Online at: `http://www.ggf.org/documents`
3. G. Allen *et al*, "Enabling Applications on the Grid: A GridLab Overview", in *International Journal of High Performance Computing Applications*, Vol. 17, No. 4, SAGE Publications, Nov. 2003, pp. 449–466.
4. G. Allen *et al*, "The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for the Grid", *Proceedings of the IEEE*, Vol. 93, No. 3, 2005, pp. 534–550.
5. K. Ballinger, D. Ehnebuske, *et al* (Eds.), *Basic Profile Version 1.1 (Final)*, The Web Services-Interoperability Organization (WS-I),Aug. 2004. Online at: `http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html`
6. R. van Engelen, K. Gallivan, "The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks", in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, IEEE Press, 2002, pp. 128–135.
7. R. van Engelen, G. Gupta, S. Pant, "Developing Web Services for C and C++", in *IEEE Internet Computing*, Vol. 7, No. 2, pp. 53–61, Mar./Apr. 2003.

8. I. Mandrichenko (Ed.), W. Allcock, T. Perelmutov, *GridFTP v2 Protocol Description*, Global Grid Forum Recommendation Document GFD.47. Online at: `http://www.ggf.org/documents`
9. J. Novotny, M. Russell, O. Wehrens, "GridSphere: a portal framework for building collaborations", in *Concurrency and Computation: Practice and Experience*, Vol. 16, No. 5, Wiley, Apr. 2004, pp. 503–513.
10. A. Rajasekar, M. Wan, R. Moore, "MySRB and SRB - Components of a Data Grid", in *Proceedings of the 11th International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, pp. 301–310, July 2002.
11. R. C. Ris, L. H. Holthuijsen, N. Booij, "A Spectral Model for Waves in the Near Shore Zone", in *Proceedings of the 24th International Conference on Coastal Engineering*, Kobe, Oct. 1994, Japan, pp. 68–78, 1994.
12. J. Waldo, G. Wyant, A. Wollrath, S. Kendall, *A Note on Distributed Computing*, Technical Report TR-94-29, Sun Microsystems Laboratories, Inc., Nov. 1994. Online at: `http://research.sun.com/techrep/1994/abstract-29.html`
13. J. Westerink, R. Luettich, A. Baptista, N. Scheffner, and P. Farrar, "Tide and Storm Surge Predictions Using Finite Element Model", in *ASCE Journal of Hydraulic Engineering*, pp. 1373–1390, 1992.
14. Unidata's Network Common Data Form (NetCDF). `http://my.unidata.ucar.edu/content/software/netcdf/index.html`
15. The SCOOP Program's Coastal Model Project Website. `http://www1.sura.org/3000/3310_Scoop.html`