Grid Style Web Services for climate*prediction*.net

Daniel Goodman and Andrew Martin*

28th May 2004

Abstract

In this paper we describe a architecture which implements call and pass by reference using asynchronous Web Services. This architecture provides a distributed data analysis environment where functions can be dynamically described and used.

1 Introduction

This report outlines the architecture and rationale behind work in progress on a web service based grid service style system[3, 4] for distributed analysis of the data produced by climate*prediction*.net (CPDN)[6]. CPDN aims run hundreds of thousands of models of the climate from different starting points. These models are run in the idle time on volunteers' PCs in a similar manner to SETI@home. About 5% of each model is uploaded to one of a set of upload servers, creating a large distributed dataset. It is then desirable to be able to analyse this data without disturbing the distributed nature of the data store.

The proposed architecture for this system is based on the use of asynchronous web services. It is intended to provide dynamically describable data analysis functions over a distributed system. The complexity of this distributed system should as far as possible be abstracted away from the user. Although it has many similarities with the methodologies suggested in WS-RF and WS-GAF it is developed without strict adherence to any particular set of specifications, using only standard web services. In the fullness of time, in order to achieve interoperability with systems such as the NERC DataGrid, and when suitable tools and libraries are available we would expect to adopt some higher level standards.

The initial implementation of this system is being written in Java and deployed with Apache Axis running on Tomcat. It is being written with the hope of producing implementations in other languages eg. C and Fortran. All of these implementations will support the same set of web service interfaces and message specifications allowing interoperability between systems, and making it easier to incorporate legacy code.

2 System Architecture

2.1 Global System Principles

Because of the large size of the data set, the data is passed by reference. Functions are called by reference to enable them to be passed to other functions as arguments. The references are URIs which contain information including the URL of the service provider holding the data, the type of the data and a unique identifier in the domain of the holding service provider. Each of these references allows the lookup of a handle which contains a reference to the object in question and holds meta data about the object. This meta-data currently contains a mixture

*Oxford University Computing Laboratory

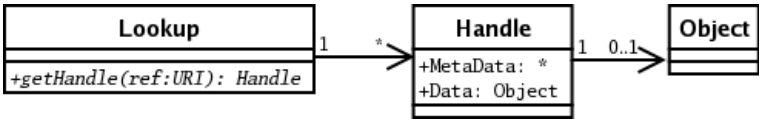

Figure 1: UML structure of Lookup and Handles.

of implementation specific information and object specific information such as if the object is being used, and if it is being used, a reference to the function using it, finally the time at which its existence can no longer be guaranteed[1]. In this implementation the handles are stored in a hash table for lookup.

Data cannot be *moved* from one service provider to another. Instead it must be *copied* and the copy given a new reference by the receiving service provider. The method for transfer can be data type dependent. In the case of matrices the technique that was chosen is to turn the file into a netCDF[5] format file and transfer this to the new location by FTP, at which point the matrix can be reconstructed. The choice to use netCDF for the matrix transfer was made because this is one of the formats that is going to be supported by the NERC DataGrid project[1, 2], and is the native format of the returned climate data. The choice of FTP was made because although netCDF has libraries which directly support the access of the data files over HTTP, by using FTP it is possible to remove the need to signal between the service providers. Figure 2 shows a generic outline of the sequence of events if a function is called using a reference to data on another service provider.[2]

2.2 System Breakdown

The system is constructed from three logical units:

Data Store Provides a set of services that can take a data description, construct an appropriate data structure and return a reference to it.

Data Processor Provides a set of services that can take arguments and perform a function on those arguments. It also provides methods for adding new functions to the service provider.

Process Coordinator Coordinates the calls to other parts of the system. This is used to make the distributed system appear as a single system.

These units can exist independently or grouped together as appropriate to the infrastructure that they are being deployed on. For instance it may be desirable to separate the data store from the numerical calculations for performance reasons. Alternatively you may want each server to provide all the services.

2.2.1 Data Store

Currently the only data structures that we are creating from the data store are matrices. The service provider receives a data descriptor, this descriptor is then parsed into an SQL query to determine which files in the results set contain the requested information, and a set of range descriptors detailing the information to be retrieved from each file. This information is then passed to a matrix factory, which creates the matrix which is placed in a handle and added to the available resources.

[1] If the is space to continue to store object, the is no reason actively to dispose of it, instead its destruction can be delayed until the garbage collector is invoked.

[2] The function call isn't restricted to being from a process coordinator.

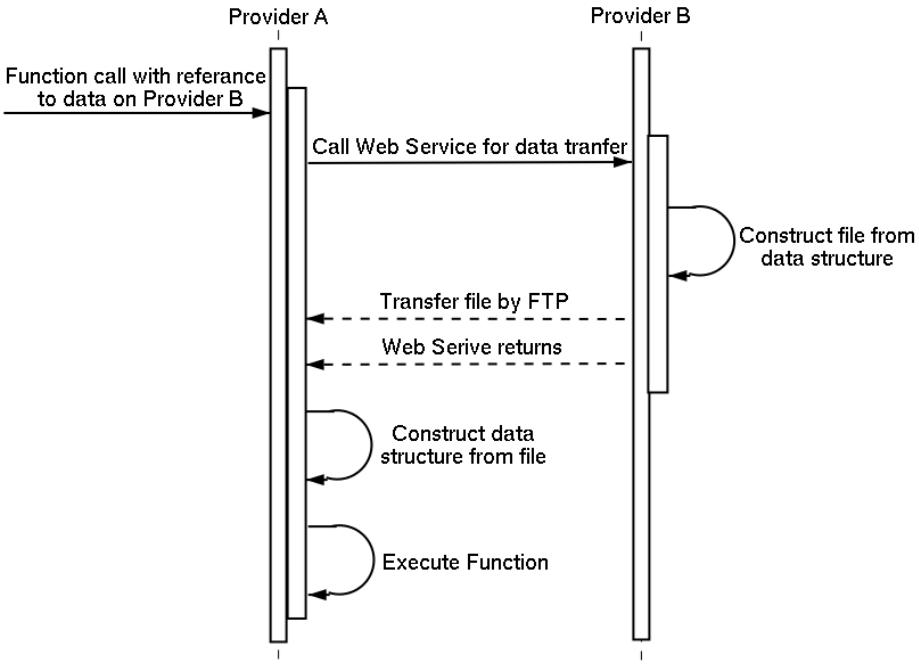

Figure 2: Process diagram of a non local reference retrieval.

2.2.2 Data Processor

Initially a Data Processor has a set of predefined functions referenced by publicly defined URIs. When one of these is called, the call has a function reference and an array of argument references. These are then resolved into a function handle and an array of argument handles respectively. The function handle holds a reference to a function constructor. This is called with the array of argument handles, and constructs a function object that will perform the function specified by the URI on the arguments. The function object is then invoked and disposed of after it returns. The advantage of this method is that it is possible to create an arbitrary number of concurrent occurrences of the same function, and trivially avoid problems with race conditions.

It is not feasible to describe all the required functions in the initial system setup, and to make repeated calls to each system would create a tightly linked impractical system. To overcome this, a little language is defined to enable new function constructors to be created, registered in the lookup and a reference returned. This language allows basic conjoins such as sequential composition, parallel composition and *if* and *while* statements to be used on existing functions. It allows for functions to be used as arguments, and for data handles to be made constant, avoiding the need to be passed into the function each time it is called. This language is implemented by first parsing the input string into a tree, and then recursively constructing the parts of this tree to create a function constructor that contains mappings from the arguments it receives, to the arguments it needs to pass to its children, and an array of handles which are constant arguments. This has the advantage that function constructors cannot become poorly defined by other functions or data becoming out of date, and being disposed of. In an implementation using a language without garbage collection, issues like this could be dealt with by adding another field to the handle meta-data and slightly more complex handle constructors and destructors. As this work matures it is hoped that this language can be transformed in to a sub-language of an appropriate workflow language.

If a function call receives a reference which refers to data on another service provider, it will copy that data across and invoke the function on this copied data. If a function reference refers

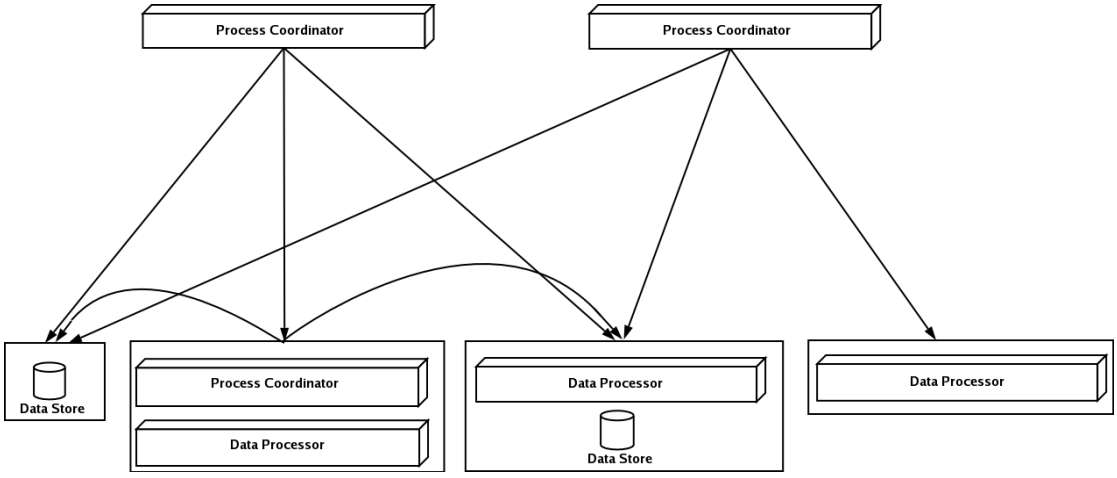

Figure 3: An example of a possible arrangement of the logical components

to a different service providers function, two options present themselves. The first is to invoke the service on the other service provider and let the data copying take care of everything. The other option is to convert the function structure back into XML and use this to make a copy of the the function locally. The second option would require all function providers to provide the same basic functions; however it would prevent moving potentially large amounts of data.

2.2.3 Process Coordinator

This process has to deal with the management of all of the other processes so that the data analysis can be implemented across the whole data set using the tools provided. It is necessary for this piece to be aware of how to divide and merge results. It is planned to implement this by having handles that hold a set of references. This will enable easy splitting of function calls on to their appropriate subsets of the data. The foreseen complications arise with the merging of the data back together again. All other requirements of this function can simply be dealt with by having a list of service providers that are managed by this service, passing down requests to them, and storing the responses.

It was stated at the beginning of this section that it is possible to separate locations of the data analysis and the data store. This is the logical unit that would be responsible for performing this function. This could be done as a static setup, or could be an adaptive algorithm monitoring system performance. The scope for invention here is almost unlimited.

A point worth noting here is that there is no requirement for there just to be one process coordinator: each service provider could potentially be a process coordinator. This, coupled with some reasonable failure recovery mechanism for the other service providers, would prevent the whole system failing in the event of any one provider failing. It is also possible to build a tree structure of these services to stop them becoming a bottle neck for the system as the number of data stores and data processors increases. This is just an example of the composite design pattern being applied.

2.3 Initialisation

When the system is initialised each process knows its own URL, and any relevant details about its own structure. In addition to this, the process coordinators knows the URLs of the data stores that they can use when generating new data structures from experimental data, and the

URLs of the data processors they can distribute commands to. From this point on all data locating is done via the URL and domain specific id in the reference URI.

3 What the future holds

Although this work is well underway there still is a lot of work to do. Other than completing the work outlined above, ideally the following will be added:

- Interfaces to allow seamlessly interaction with the NERC data grid when retrieving data to analyse.
- Data transfer currently requires large amounts of transient space. The file is fully constructed, then transfered, used and finally destroyed. This could be avoided by a virtual file construct. It will be interesting to see how this is dealt with on the NERC DataGrid when they support custom netCDF files generated from their stored data.
- Type checking for submitted functions would help detect errors before to much computing power has been wasted on functions that will fail to return correctly.
- Security and user privileges will need to be addressed before this system can be used by the wider world. The current intention is to do this via an additional part of the reference URI and a set of handlers that the requests have to pass through before they are executed. This holds well with the Apache Axis handler control structure.

4 Conclusion

Here we have shown the basis for a grid style architecture constructed out of stable web service specifications[3] and adaptations of design patterns. This architecture enables functions to be dynamically deployed and run concurrently in a safe yet not excessively restrictive environment. The use of call and pass by reference functions makes for a clean and simple divide between the data stores and the numerical calculations allowing for automatic dynamic restructuring. This is abstracted away from the user, and a simple well defined public interface is provided. Because of the loosely coupled rough grained structure between service providers, the performance is expected to be similar to that of the code for the functions being run in their native environment.

Acknowledgements

The authors would like to thank Carl Christensen and everyone else who has helped with this project. This work is funded by NERC.[4]

[3] SOAP & WSDL

[4] Ref:NER/S/G/2003/11992

Appendix

An out line of the service interfaces for each logical unit.

Common to all

```
copy(URI source, URI target)
createHandle(String grade, String type)
discard(URI ref)
sendData(String remoteLocation, URI source, URI signalRef)
signal(String ref)
```

Data Store Services

```
createMatrix(String[][] files, RangeDescriptor[][] values, String grade):URI
```

Data Processor Services

```
createFunction(StmtTree s, String grade):URI
executeFunction(URI funct, URI[] args)
```

Processor Coordinator Services

```
createFunction(String funct, String grade):URI
createMatrix(String dataDescription, String grade)
executeFunction(URI funct, URI[] args)
```

References

[1] Bryan Lawrence, David Boyd, Kerstin Kleese van Dam, Roy Lowry, Dean Williams, Mike Fiorino, and Bob Drach. The NERC DataGrid. Technical report, CCLRC - Rutherford Appleton Laboratory, 2002.

[2] Bryan Lawrence, Ray Cramer, Marta Gutierrez, Kerstin Kleese van Dam, Siva Kondapalli, Susan Latham, Roy Lowry, Kevin O'Neill, and Andrew Woolf. The NERC DataGrid prototype. Technical report, CCLRC e-Science Centre, British Atmospheric Data Centre and British Oceanographic Data Centre, 2003.

[3] Savas Parastatidis, Paul Watson, and Jim Webber. Grid resource specification. Technical report, North East Regional e-Science Centre, 2003.

[4] Savas Parastatidis, Jim Webber, Paul Watson, and Thomas Rischbeck. A grid application framework based on web services specifications and practices. Technical report, North East Regional e-Science Centre, 2003.

[5] Russ Rew, Glenn Davis, Steve Emmerson, and Harvey Davies. *NetCDF User's Guide*. UCAR/Unidata Program Center P.O. Box 3000 Boulder, Colorado, USA 80307, 2.4 edition, Feb 1996.

[6] David Stainforth, Jamie Kettleborough, Andrew Martin, Andrew Simpson, Richard Gillis, Ali Akkas, Richard Gault, Mat Collins, David Gavaghan, and Myles Allen. Climate*prediction*.net: Design principles for public-resource modeling research. In *14th IASTED International Conference Parallel and Distributed Computing and Systems*, Nov 2002.