INTERACTIVE DATA ANALYSIS ON THE GRID USING GLOBUS 3 AND JAS3

A.S. Johnson, V. Serbo, M. Turri, SLAC, Stanford University, Stanford, California 94309,USA
B. Anathan, D.A. Alexander , Tech-X Corp., 5621 Arapahoe Ave, Boulder, CO 80303, USA

Abstract

This paper provides a status report on the Dataset Analysis Grid Service (DAGS) project. The aim of the service is to allow fully distributed analysis of large volumes of data while maintaining true (sub-second) interactivity. The Grid related components are based on OGSA style Grid services, and to the maximum extent possible uses existing Globus Toolkit (GT3) services. All transactions are authenticated and authorized using GSI (Grid Security Infrastructure) mechanism - part of GT3. JAS3, an experiment independent data analysis tool is used as the interactive analysis client.

INTRODUCTION

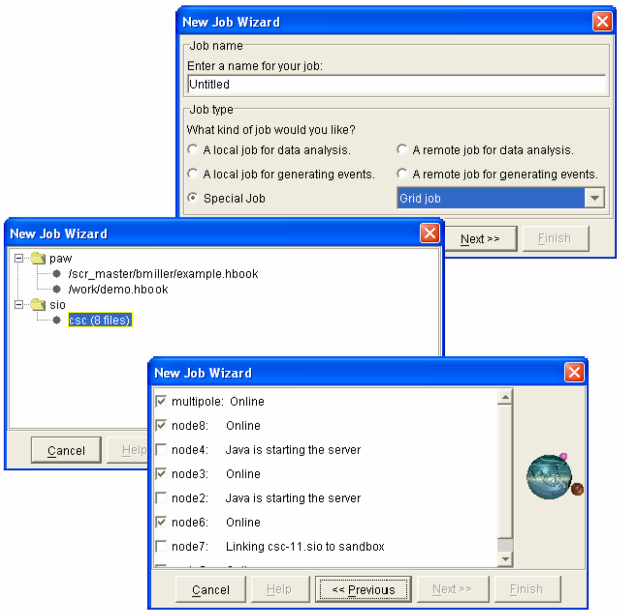
In this paper we describe a toolset for performing interactive data analysis on the Grid [1]. The goal of the project is to allow data analysis to be performed in parallel on a set of worker machines, to maximize the amount of CPU power and IO capability that can be delivered in a short time, while presenting results to the user with a graphical client running on their desktop workstation. Unlike most Grid projects which aim at batch style submission of data analysis, where turnaround time is typically from minutes to hours, our goal is to provide sub-second response times, for example allowing cuts to be changed dynamically and the resulting change in plots to be seen immediately, or to allow plots to be updated dynamically as a longer analysis task is progressing. Ideally the details of how the analysis is run on the Grid should be hidden from the user; we want them to have the impression that their desktop machine has seamlessly become capable of analysing more data in a given time.

PREVIOUS WORK

In an earlier phase of this project we took the existing JAS2 [2] graphical data analysis system and extended it to work in a distributed environment. With these extensions JAS2 was able to work in three modes:

1. *Local mode*, where data to be analyzed, analysis algorithms, and graphical user interface (GUI) all run on the users desktop.
2. *Client-Server mode*, where data to be analyzed lives on a remote server, and analysis code is sent to the server to be run, but results are presented to the user using their desktop GUI.
3. *Distributed mode*, where data and analysis are distributed to a set of worker machines, where the analysis runs in parallel, but results are still presented on the user's desktop

In this third mode, we use Globus 2 [3] to start services on each of the worker nodes, and use Java Remote Method Invocation (RMI) as the network protocol for communication between the desktop machine and the worker node. Communication between the client and workers is mediated by a gateway node that combines results from each worker machine and presents the combined result to the client. Figure 1 shows the components added to the JAS2 GUI to enable the distributed mode.


New Job Wizard

Job name
Enter a name for your job: Untitled

Job type
What kind of job would you like?
( ) A local job for data analysis. ( ) A remote job for data analysis.
( ) A local job for generating events. ( ) A remote job for generating events.
(•) Special Job — Grid job

paw
/scr_master/bmiller/example.hbook
/work/demo.hbook
sio
csc (8 files)

[✓] multipole: Online
[✓] node8: Online
[ ] node4: Java is starting the server
[✓] node3: Online
[ ] node2: Java is starting the server
[✓] node6: Online
[ ] node7: Linking csc-11.sio to sandbox

Figure 1: The JAS2 wizard for creating a Grid job prompts the user for their choice of data files to analyze and then shows the user the status of the launch of the servers on each of the worker nodes.

CURRENT PROJECT

The current project builds on the earlier work performed with Globus 2 and JAS2, but attempts to standardize on OGSA [4] web services, reusing standard services provided by Globus 3 and others where they exist, and implementing new services where necessary. Considerable thought has been put into decoupling services as much as possible, so that individual services can be independently reusable.

In keeping with modern practice we have kept a clear separation between the definition of the public interfaces and the implementation of those interfaces. The interfaces have been designed in consultation with the PPDG-CS11 group [5], while the reference implementation, which we call Dataset Analysis Grid Service (DAGS), has been developed by the authors.

Dataset Catalog Service

The first interface designed as part of the current project was the Dataset Catalog Service (DCS). The intention of this interface is to provide a public API for use by physicists for finding data samples of interest. This API can be used directly or indirectly via some graphical or console based search tool. The interface allows datasets to be arranged hierarchically so that they can be browsed (like files in a file system), but also allows datasets to have associated meta-data and provides a mechanism for the user to perform searches based on the meta-data.

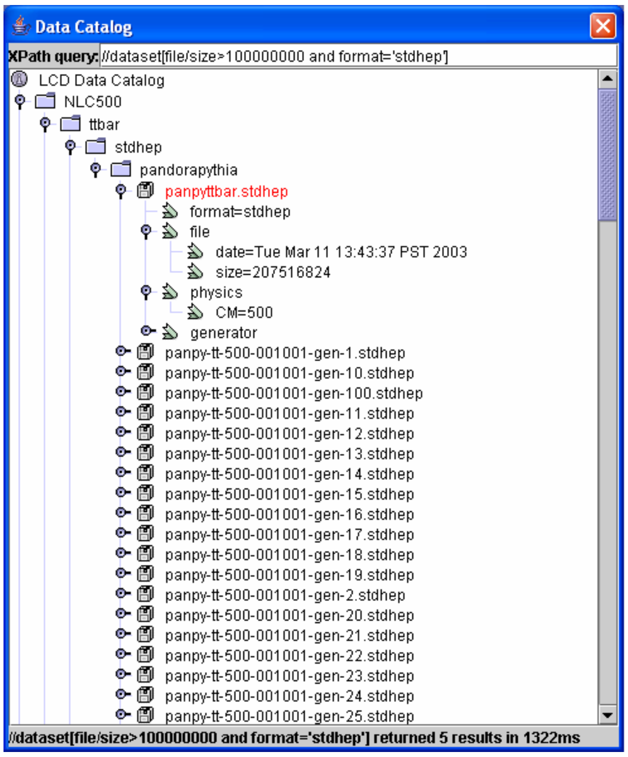
The interface does not make any assumptions about how the datasets are stored, or how the meta-data is stored, the explicit intention is to make it possible to implement this catalog interface on top of any existing dataset catalog, and thus to allow tools to search many different dataset catalogs without having to have a custom interface to each catalog. In order to provide maximum flexibility, the result of a search is a list of DatasetID's, each consisting of an opaque string ID, and a Grid Service Handle of a service which is able to interpret the ID and make the data available for further analysis.

In addition to providing the definition of the DCS interface we have produced a reference implementation which uses a simple XML description of the available datasets and meta-data, and provides a Java GUI for browsing or searching for datasets (Figure 2). The dataset query language supported by the reference implementation is XPath [6].

Dataset Analysis Grid Service

The Dataset Analysis Grid Service (DAGS) builds upon the design concepts of the Dataset Catalog Service to build a complete distributed interactive dataset analysis system. DAGS aims to allow a client to perform data analysis in parallel on a farm of machines, to improve data throughput, while presenting the results to the user through a GUI which hides as much as possible of the distributed nature of the system.

DAGS requires that a set of custom grid services are installed on a single gateway node at each site. The only initial requirements on the nodes used for data analysis (the worker nodes) are that Globus Toolkit and a Java VM be installed; everything else is deployed dynamically by DAGS. There is no requirement that the worker nodes have access to the internet, all communication with the client takes place via the gateway node. On the client node the only requirement is that JAS3 is installed, no Grid software is required on the client node except for the Java CoG kit [7] which is dynamically deployed by DAGS as required using the JAS3 plugin mechanism.


Data Catalog
XPath query: //dataset[file/size>100000000 and format='stdhep']

```
LCD Data Catalog
  NLC500
    ttbar
      stdhep
        pandorapythia
          panpyttbar.stdhep
            format=stdhep
            file
              date=Tue Mar 11 13:43:37 PST 2003
              size=207516824
            physics
              CM=500
            generator
          panpy-tt-500-001001-gen-1.stdhep
          panpy-tt-500-001001-gen-10.stdhep
          panpy-tt-500-001001-gen-100.stdhep
          panpy-tt-500-001001-gen-11.stdhep
          panpy-tt-500-001001-gen-12.stdhep
          panpy-tt-500-001001-gen-13.stdhep
          panpy-tt-500-001001-gen-14.stdhep
          panpy-tt-500-001001-gen-15.stdhep
          panpy-tt-500-001001-gen-16.stdhep
          panpy-tt-500-001001-gen-17.stdhep
          panpy-tt-500-001001-gen-18.stdhep
          panpy-tt-500-001001-gen-19.stdhep
          panpy-tt-500-001001-gen-2.stdhep
          panpy-tt-500-001001-gen-20.stdhep
          panpy-tt-500-001001-gen-21.stdhep
          panpy-tt-500-001001-gen-22.stdhep
          panpy-tt-500-001001-gen-23.stdhep
          panpy-tt-500-001001-gen-24.stdhep
          panpy-tt-500-001001-gen-25.stdhep
```

//dataset[file/size>100000000 and format='stdhep'] returned 5 results in 1322ms

Figure 2: The interface for the dataset catalog service. This figure shows how users can browse the catalog, or perform a meta-data query, resulting in the selected datasets being highlighted.

Figure 3 shows the conceptual architecture of the current DAGS system. Although the intention is to switch to using OGSA interfaces throughout, currently the system uses Java RMI in places, largely for historical reasons. The intention is to remove all RMI interfaces in future (but see the comments on performance below).

Conceptually the system works as follows: The user first uses the JAS3 proxy login plugin to authenticate to the Grid and obtain a proxy certificate. The user then chooses one of more datasets to be analysed, using the data chooser plugin in JAS3 (based on the Dataset Catalog Service described earlier). The DAGS client is then used to submit analysis tasks to the system which work on the selected datasets. The analysis tasks can be specified as scripts (currently Pnuts [8] and Python [9] scripting languages are supported) or as compiled Java code.

The Dataset Analysis Manager Service uses the DatasetID's sent be the user to locate the data using the dataset locator service. The dataset locator service returns a handle to a data splitter service, which knows how to partition the data to be analysed into smaller parts which can be analysed in parallel. Depending on the specific type of data, the data splitter service may actually create many small files containing subsets of the data, or it may know that the dataset already exists as many component files, or it may simply logically partition the data, for instance by choosing different sets of records from a database. The Dataset Analysis Manager Service then chooses a set of worked nodes to be used for the analysis, and uses the Globus Reliable File Transfer Service to move the data and any required programs to the worker nodes. An Analysis Server is then started on each worker node, using the Globus Managed Job Service and the analysis script or program provided by the user is dynamically loaded into the analysis server.

As the analysis executes various results may be produced, such as histograms, n-tuples, log files, *etc*. These are fed via the Result Merging Service back to the client. If the user chooses to view one of these results while the analysis code is still executing, for example a histogram, they will see the resulting plot update in real time, as the merging service will (on request) continually get the result components from each worker node, combine them together, and feed them back to the client.

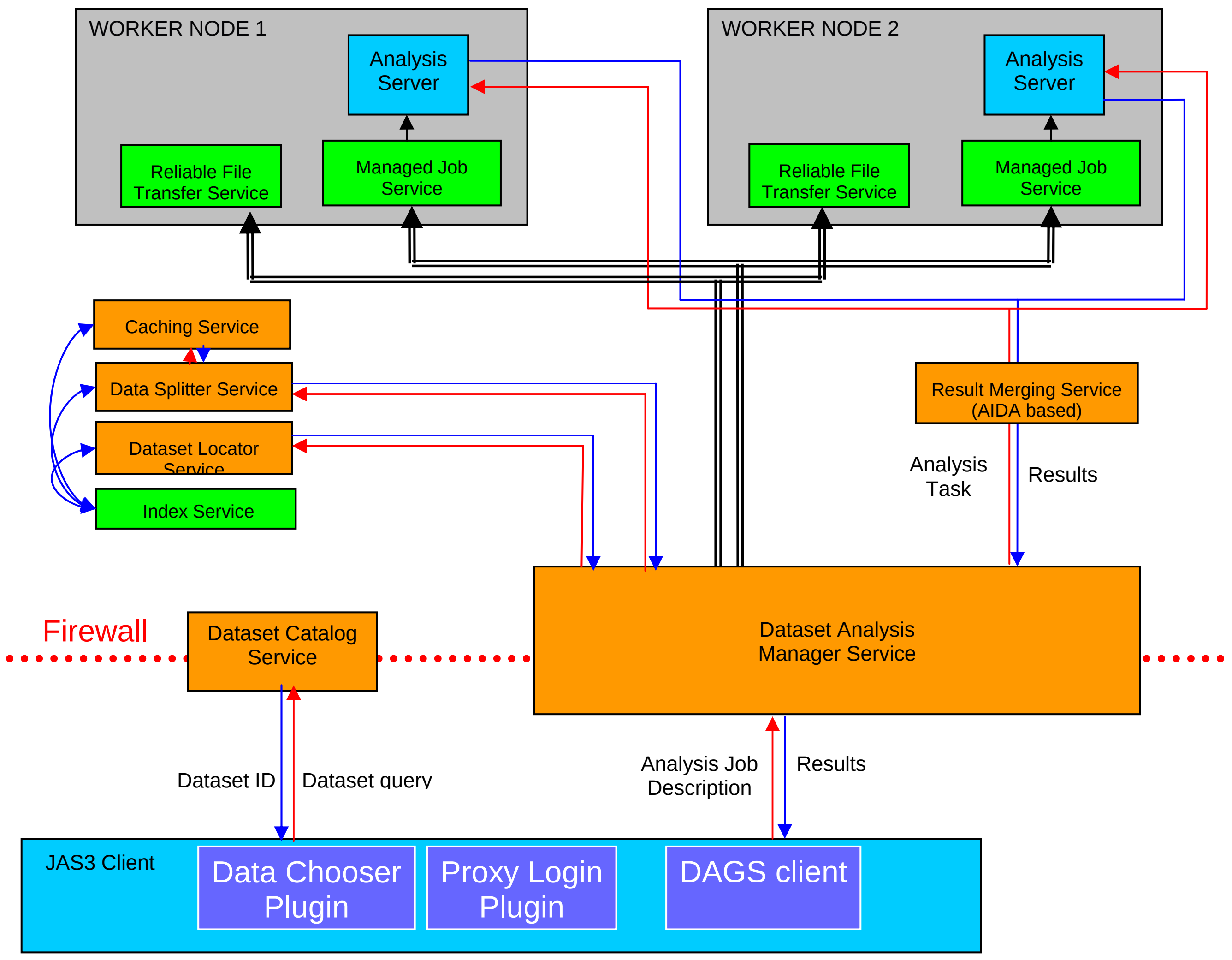


Figure 3: Conceptual diagram of the components comprising the Dataset Analysis Grid Service (DAGS).

STATUS AND PERFORMANCE

The DAGS system is currently installed as a prototype on a small number of nodes at SLAC. One issue currently being worked on is the performance of Globus OGSI compared to Java RMI used earlier in the project. Some results on performance for trivial operations are shown in table 1. The times shown in the table are the total elapsed times for 100 remote method calls over a 100Mbps LAN, excluding the first call. For Globus we used Globus Toolkit 3.2 running with the Globus Grid service container. The test code is derived from the Auction service example distributed with the Globus toolkit. For RMI we used Java 1.4.2 with an implementation functionally equivalent to the auction service used for the Globus tests.

Possible causes for the relatively slow performance of Globus is the overhead of marshalling and unmarshalling XML messages, the overhead associated with GSI security, and the overhead of establishing an http connection for each message. We are currently working with members of the Globus team to try to fully understand the overhead associated with OGSI, and hopefully rectify or workaround the problem.

Table 1: Comparison of elapsed time for invoking a trivial remote method call using different protocols. Each time quoted is for invoking the remote method 100 times over a 100Mbps LAN.

RPC Protocol	**Elapsed Time**
Java RMI	96 ms
Globus 3.2 (non-secure)	22 seconds
Globus 3.2 (secure)	112 seconds

Once the performance problems have been ironed out we plan to deploy the system with real users and real data. Our initial plans are to target users doing simulation studies for a future International Linear Collider. In addition we are exploring interoperability with other systems, in particular Clarens [10] and gLite [11].

ACKNOWLEDGEMENTS

Work supported by Department of Energy contracts DE-AC03-76SF00515 and DE-FG02-03ER83856.

REFERENCES

[1] I.Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure. Morgan Kauffmann Publishers, 1998 (2nd edition, Nov 2003).
[2] http://jas.freehep.org/
[3] The Globus Toolkit, http://www.globus.org/
[4] Open Grid Services Architecture (OGSA) http://www.globus.org/ogsa/
[5] CS11 is working group of the Particle Physics Data Group (PPDG), http://www.ppdg.net/pa/ppdg-pa/idat/index.html
[6] XML Path Language (XPath) 2.0. W3C Working Draft 29 October 2004. http://www.w3c.org/TR/xpath20/
[7] Java CoG Kit: http://www-unix.globus.org/cog/java/
[8] Pnuts scripting language: https://pnuts.dev.java.net/
[9] Python programming language: http://www.python.org/
[10] C. Steenberg, The Clarens Grid-enabled Web Services Framework: Services and Implementation, presented at this conference.
[11] gLite, Lightweight Middleware for Grid Computing: http://glite.web.cern.ch/glite/