SLAC-PUB-12289

January 2007

Framework for Interactive Parallel Dataset Analysis on the Grid

David A. Alexander, Balamurali Ananthan
Tech-X Corporation
5621 Arapahoe Ave, Suite A
Boulder, CO 80303
{alexanda,bala}@txcorp.com

Tony Johnson, Victor Serbo
Stanford Linear Accelerator Center
SLAC, Mail Stop 71
Stanford, CA 94309
{tony_johnson,serbo}@slac.stanford.edu

Abstract

We present a framework for use at a typical Grid site to facilitate custom interactive parallel dataset analysis targeting terabyte-scale datasets of the type typically produced by large multi-institutional science experiments. We summarize the needs for interactive analysis and show a prototype solution that satisfies those needs. The solution consists of desktop client tool and a set of Web Services that allow scientists to sign onto a Grid site, compose analysis script code to carry out physics analysis on datasets, distribute the code and datasets to worker nodes, collect the results back to the client, and to construct professional-quality visualizations of the results.

1 Introduction

Part of what could be considered hype associated with Grid Computing [1] is the promise that shared resources can be easy to use. For the user who wants to interactively analyze data on a Grid with the current version of the Grid middleware, this is only true in a limited sense. Once middleware is properly installed on the target resources, and once the user has the proper credentials, and once the user is properly recognized by the Virtual Organization (VO) [2], then it is true that the user can run grid jobs invoking any installed application. To fully meet the expectations stirred by promotional publicity that promise services such as being able to run truly custom and interactive processes, out-of-the-box Grids such as the Open Science Grids [3] or EGEE Grids [4] need certain modifications.

We define interactivity as the ability to analyze a dataset and give back partial results on time scales of less than a minute. This is opposed to batch processing where jobs that may take many minutes or hours are run without any feedback to the user until the jobs are finished. By interactivity, we also mean that the user can change their analysis algorithms on the fly, implying the need for controls to stop and restart an analysis that is in progress. Furthermore, with respect to datasets, this means that the user must be able to easily select the dataset to be analyzed and change the dataset during the analysis session, all while interacting with the dataset by name or metadata rather than on a file basis. To accomplish this, a system that provides interactive dataset analysis must be in close contact with the client and be able to handle datasets in a sophisticated way.

Modern physics experiments and simulations like the Large Hadron Collider (LHC) [5] and International Linear Collider (ILC) [6] produce many terabytes of data each year. These large datasets can be analyzed in a batch mode that requires no interactivity. But in many cases interactivity in terms of custom analysis is necessary to fine tune an analysis that may eventually become a production batch analysis. The process of fine tuning or custom analysis requires a system that allows rapid development and re-running of an analysis while making incremental changes. Since researchers need large datasets to produce statistically meaningful results, the ability to take advantage of grid processing power while performing interactive analysis may be essential to advance the science.

We must make the distinction that what we refer to as parallel analysis in this paper involves the perhaps more simplified case where the data is record or event based and the same analysis is to be performed on each event. This kind of parallelism is not the same as the parallelism referred to in codes that use some sort of message passing interface to heavily communicate between the processes of the analysis. Our framework is targeted to datasets that can be split and where the analysis results can be logically merged. Examples of applications that require this capability include analysis of particle collider events in high energy physics, DNA sequencing combinations in cellular biology, and stock trading records in business.

Contributed to 2006 International Conference on Paralle Processing Workshops (ICPPW 06), 14-18 August 2006, Columbus, Ohio

Work supported in part by the US Department of Energy contract DE-AC02-76SF00515

This type of parallel analysis on the Grid can still provide much needed computation power for large dataset analysis. With a greater number of processors available on the Grid, the process of repeatedly running the analysis over the same dataset and fine tuning the analysis code consumes much less time than when it is analyzed locally on a single processor machine. The question is: how easy it can be to take advantage of such Grid power if the proper services are offered?

We present here a framework that performs interactive parallel dataset analysis on the Grid. The framework contains a set of Web Services that allows users to export analysis code that they have tested on their local data to run on a Grid site associated with their VO. The framework provides capability to create an interactive analysis session, choose a dataset from a catalog service to be staged onto the worker nodes for parallel analysis, upload their custom analysis code to the Grid, and get back intermediate results as they are produced. The key additional requirements to the standard Grid are a dedicated timely scheduler queue and a mechanism for communication from workers to the client.

2 Interactive Parallel Analysis Framework

What we describe in this section is a framework and the general requirements for that framework that we have determined are necessary to provide interactive parallel analysis. We call this the Interactive Parallel Analysis (IPA) framework. Figure 1 shows the diagram of the IPA framework parts for interactive parallel dataset analysis on the Grid.

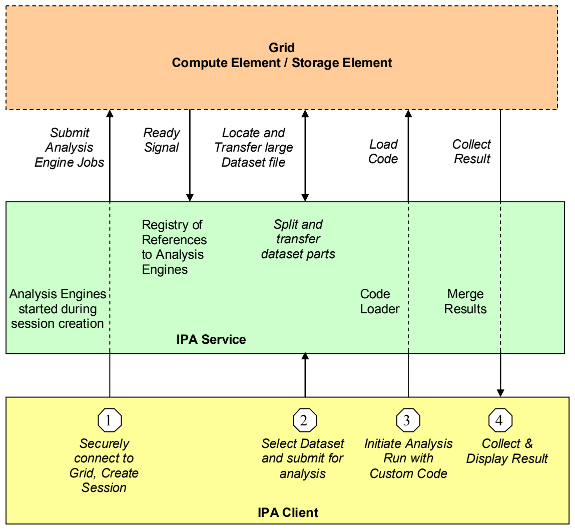

Grid
Compute Element / Storage Element
Submit Analysis Engine Jobs
Ready Signal
Locate and Transfer large Dataset file
Load Code
Collect Result
Registry of References to Analysis Engines
Split and transfer dataset parts
Analysis Engines started during session creation
Code Loader
Merge Results
IPA Service
1
Securely connect to Grid, Create Session
2
Select Dataset and submit for analysis
3
Initiate Analysis Run with Custom Code
4
Collect & Display Result
IPA Client

Figure 1. Framework for Interactive Parallel Analysis (IPA) on Grid

We envision the framework as three layers: The client layer, the service layer, and the Grid layer. The user interacts with the client layer and follows four steps to analyze a dataset. Within the service and Grid layers we have introduced a concept of an analysis engine. Analysis engines are processes that accept a dataset and an analysis script and analyze the dataset using the script to produce a result.

In the following sections, we describe the requirements for the Grid, capable of analyzing large datasets.

2.1 Need for a Dataset Catalog

To start with, the user will need some way of choosing the dataset that is to be analyzed. An abstract metadata catalog of datasets would be an ideal solution for this. The metadata contains all the information about the datasets, but does not contain the actual data itself. The metadata should be organized in a hierarchical fashion where the user can browse the catalog and choose the dataset of interest. An added advantage would be if a dataset or set of datasets could be searched based on a query pattern.

2.2 Need for High-Level Dataset Handlers

As mentioned in section 2.1, what is chosen by the user from the catalog is a pointer to the actual dataset. Some mechanism is needed to resolve the chosen metadata to the actual dataset, as well as a mechanism to stage the dataset for analysis. In effect, we need a Locator and a Splitter. The Locator will take the dataset identifier and resolve it to the actual location of the dataset. The Splitter will split the selected dataset and disperse it over the machines in the Grid. The maximum number of analysis engine nodes that could be started on the Grid is determined by the Grid-VO policy.

2.3 Need for a Mechanism That Quickly Starts the Analysis Engines on the Grid

Analysis of the dataset on the Grid is performed by the analysis engines that are started dynamically on the worker machines on the Grid. The worker machines are the nodes on the Grid where the analysis would happen. This analysis engine should be started relatively quickly - within the limits of human tolerance. The analysis engines should not be statically running at all times, but instead should be started for each session and be shutdown at the end of a session. This saves computational resources on the Grid and allows the analysis engines to dynamically pickup new data format readers.

2.4 Need for a Mechanism to Stage Code for Analysis

Once the analysis engines are ready for performing analysis on the dataset, we need a way to ship the analysis code that does this analysis from the client machine to the Grid machines. The analysis code will be written by the physicists, which should take the records of the dataset as input and run the analysis.

2.5 Need for a Mechanism to Merge and Display the Results

As the analysis is performed, intermediate results should be collected from the analysis engines on the Grid and should be presented to the client in an appealing way. Getting the intermediate results quickly and presenting them in the format desired by the user is a very important requirement of this framework.

The component that performs the merging and displaying of analysis results will become a bottleneck if there are a large number of users. The system should be adaptable in such situations by being able to accommodate a sub-level of components that performs the merging and displaying of the result. This way, the workload could be distributed to the lower level of components.

3 Reference Implementation of IPA

To demonstrate the feasibility of the framework, we have built an implementation for one use case. Our implementation was fully tested with physics data from simulations of the future Linear Collider Experiment and services hosted at Stanford Linear Accelerator Center (SLAC).

Figure 2 shows the architecture of our reference implementation from the client at one end to the Grid submission site at the other end.

The Grid resources are accessed by the client through a set of Web Services that are running on a broker node on the Grid that we call a 'Manager Node'. All of the manager services are Web Services written in Java and hosted in a Globus Toolkit 4.0 [7] container, but they can also be hosted in any other container that implements the SOAP protocol [8].

We show how the client interacts with the control services, chooses the dataset, stages the dataset, loads and runs the analysis code, and finally merges and presents the result to the user in the following sections. There are two types of communication shown in the architecture diagram; the thick green arrows represent Grid calls and the thin black arrows represent Java Remote Method Invocation (RMI) calls. Architecture that we present here is ideal for custom data analysis on the Grid.

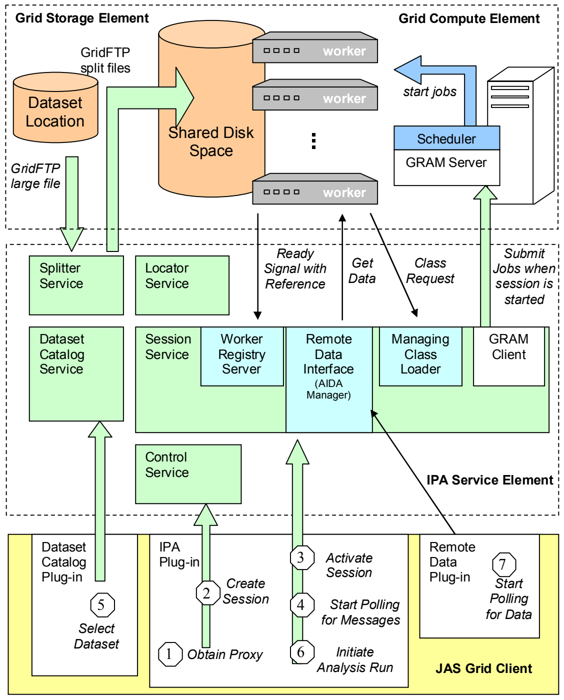


Figure 2. Architecture of IPA reference implementation.

3.1 Client

We have built a client by modifying the Java Analysis Studio (JAS) version 3.0 [9]. JAS provides a rich graphical client interface that is used to develop analysis code, navigate through datasets and graphic objects, and to graphically display the results of the analysis. The JAS client application was enhanced with three plug-in modules that communicate with the Web Services by making calls to them.

In order for the client to contact the IPA service and make Web Service calls, it first needs to mutually authenticate with the Web Service using a Grid credential. For this purpose, a Grid proxy plug-in is available on the JAS Grid client that creates a proxy certificate that can be used to authenticate the client with the service; the service could then authorize the client to use certain resources, depending on the policy of the Grid site. Once the authentication and authorization has succeeded a session is created on the session service; all calls made from the client to the Grid happen in the context of this session.

3.2 Interacting with the Control Services

At the heart of the system design is the Interactive Parallel Dataset Analysis Session Manager Service (or simply the session service). The session service creates a session for each dataset analysis. A dataset can only be analyzed in the context of this session. In addition to the session service, there are other services at the manager layer that are partly Grid-based (IPA manager service) and partly RMI based (RMI Manager). We will examine the list of components that forms this manager service in detail in the following sections.

The client is authorized and authenticated by the control service using the proxy that was created by the client. Similarly, the client authenticates the service for its validity using the mutual authentication mechanism that happens when the client initially contacts the Web Service. The control service creates an instance of session service and returns the 'pointer' to this instance to the client. Since Web Services are stateless, creating an instance of a Web Service means creation of an instance of Web Service 'resources' that can be accessed and operated by this Web Service. Globus Toolkit™ (version 4.0) implements Web Service Resource Framework (WSRF) [10] that provides the details on how to record the intermediate states of an instance of a Web Service in the Web Service resources.

When a session is started by the session service a set of analysis engines is started on the machines in the Grid where the analysis will be performed. The analysis engines are started using the GRAM server [11] that is provided as part of a standard Globus software base installation for a Grid site. The GRAM server places the request to start a pre-configured number of analysis engines on the job scheduler. The number of nodes is determined by the Grid site policy that is pre-configured on the manager service. Once the analysis engines are started on the Grid, the number of analysis engines started for this session is remembered in the session service in the session service resource.

3.3 Choosing the dataset

The dataset is chosen from the Dataset Catalog Service (DCS). The dataset catalog service is a Web Service that allows us either to browse for an interesting dataset, or to search for interesting data using a query language that operates on the metadata.

The Catalog makes no assumptions about the type of metadata stored in the catalog except that the metadata consists of key-value pairs stored in a hierarchical tree. Figure 3 shows a screen capture of the dataset chooser dialog window that appears to the user when they choose to add a dataset to a session.

3.4 Staging the Dataset

The dataset reference that is selected from the dataset catalog service contains an 'identifier' that uniquely identifies the dataset in the catalog. This dataset must be submitted to the locator service that will resolve the location of the dataset from the dataset identifier. The location could be a URL to an FTP server or a set of contiguous records in a database server.

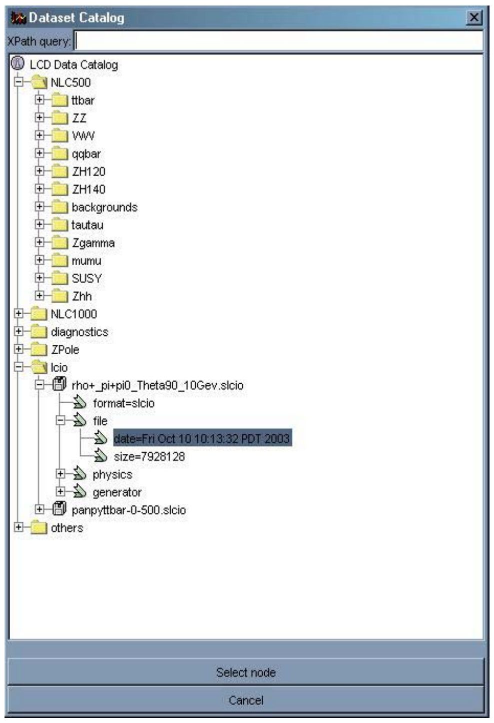


Figure 3. Dataset Catalog

In addition to the location of the dataset, the locator service returns the location of the splitter service, which is used to split the dataset. The splitter service will import the dataset from the actual location and split it into a pre-configured number of approximately equal parts. The number of parts that the dataset is split into depends on the number of analysis engines started by the session service on the Grid where analysis of dataset will take place. Once the dataset is split through the splitter service, the individual parts of dataset will be transferred using Grid FTP protocol to the analysis worker nodes.

3.5 Staging the Analysis Code

The analysis of the dataset is performed by user who provided analysis code. Our implementation currently supports Java classes and PNUTS [12] scripts. However, the framework could easily be used to support other languages. For example, C/C++ could be supported with a macro-type interpreter such as the ROOT framework [13]. The analysis engine is capable of reading the dataset that was previously staged and supplying the records from the dataset to the analysis code. The analysis code accepts the records from the dataset and processes the data to produce results.

3.6 Running the Analysis

Once the dataset and the analysis code are staged, the analysis is ready to be started. Controls are provided for users that allow them to run, pause or stop the analysis at any instant, as well as rewind to start the analysis from the beginning. After every iteration of the analysis, changes can be made in the analysis code and the new analysis code can be dynamically reloaded and used to reprocess the same dataset.

3.7 Merging and presenting the results

The sample analysis code that we used for processing our dataset in our implementation generates histograms as output. Figure 4 shows a screen capture of the client with these resulting histograms in the upper right panel.

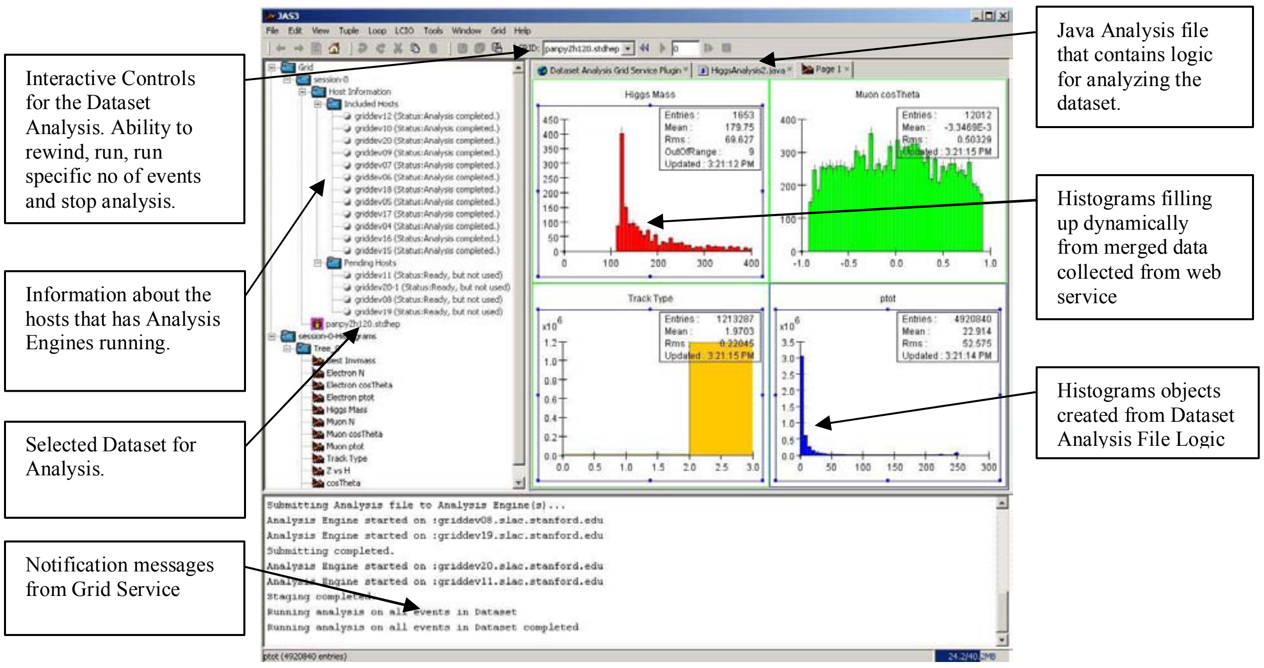
For our implementation we used the already available Abstract Interfaces for Data Analysis (AIDA) [14] – a language independent analysis toolkit that has implementations in C++, Java and Python. The analysis code makes use of the Java AIDA APIs and generates histograms from the datasets.

As soon as the analysis begins, the intermediate results from each individual analysis engines are collected and merged at the Manager node by a special manager service called the AIDA manager service. A separate plug-in on the JAS client constantly polls the AIDA manager with RMI calls to check for any updated histograms. All of the RMI connections are insecure, but we have implemented the system in such a way that none of the RMI objects could be instantiated without first creating a secure session with the Web Service.

4 Discussion

In order to show the effectiveness of analyzing datasets through our Grid system as compared to using a local system, we collected some anecdotal performance data. We took a sample analysis, a Java algorithm that looks for Higgs Bosons in simulated Linear Collider data and ran this on a dedicated 16-node Open Science Grid (OSG) queue at SLAC. The results are presented in Tables 1 and 2 below.

This simple examination of the times of various steps in the process shows a good comparison of where the bulk of the time is spent. For the local case, most of the time is spent to download large datasets and to execute the analysis on one processor. For the Grid case, most of the time is spent in splitting and moving the dataset. Moving the dataset is faster for the Grid


Interactive Controls for the Dataset Analysis. Ability to rewind, run, run specific no of events and stop analysis.
Information about the hosts that has Analysis Engines running.
Selected Dataset for Analysis.
Notification messages from Grid Service
Java Analysis file that contains logic for analyzing the dataset.
Histograms filling up dynamically from merged data collected from web service
Histograms objects created from Dataset Analysis File Logic
JAS3
Higgs Mass
Muon cosTheta
Track Type
ptot
Dataset Analysis Grid Service Plugin
HiggsAnalysis2.java
Page 1
ptot (4920840 entries)

```
Submitting Analysis file to Analysis Engine(s)...
Analysis Engine started on :griddev08.slac.stanford.edu
Analysis Engine started on :griddev19.slac.stanford.edu
Submitting completed.
Analysis Engine started on :griddev20.slac.stanford.edu
Analysis Engine started on :griddev11.slac.stanford.edu
Staging completed.
Running analysis on all events in Dataset
Running analysis on all events in Dataset completed
```

Figure 4. Screenshot of JAS with updated histograms

Table 1. Comparison of time taken for sample dataset analysis for local case vs. on the Grid.

	Local	Grid (16 nodes)
Get dataset (dataset size: 471 MB)	32 mins (over WAN)	-
Stage Dataset (download whole dataset + splitting + dataset parts transfer)	-	174 s (over LAN)
Stage Code (bytecode size: 15 kb)	-	7 sec
Analysis	13 min	258 s
Total time	**45 mins**	**4 min 19 sec**

case because the movement is over a local area network instead of a wide area network. Without looking at the scaling issues, there is a clear advantage on the Grid case when the dataset is large.

Interesting results were observed by varying the number of Grid machines available which is presented in Table 2.

Table 2. Comparison of time to stage and analyze a dataset by varying the nodes available on the Grid.

Number of Nodes	Dataset (471MB) Stage Time			Analysis Time
	Move Whole	Split	Move Parts	
1 node	63 s	120 s	105 s	330 s
2 nodes	63 s	120 s	77 s	287 s
4 nodes	63 s	115 s	70 s	190 s
8 nodes	63 s	117 s	65 s	148 s
16 nodes	63 s	124 s	50 s	78 s

The staging scaling is complicated and somewhat counter-intuitive. The splitting varies little with the number of nodes, because the splitter must iterate through the entire dataset in all cases and only has a very small input/output overhead for the number of split files. Moving the split files has overhead that will increase with the number of target files, but the transfers are done in parallel. The result is that the time taken slightly decreases as the number of nodes increases.

The analysis scaling is fairly straightforward and decreases with the number of processors. The processing time as compared to the local case is not 1/16th because the local processor was 1.7 GHz and the Grid processors were 866 MHz.

The following equations are fitted from the above measurements where T is the time in seconds, X is the dataset size in MB, N is the number of compute nodes, and where we have used 5.3 seconds as a standard time to run our sample Higgs Boson calculation on a 1 MB dataset. The final equations are given in terms of X and N. For the local case, we have

$$T_{Local} = T_{move} + T_{analyze}$$
$$T_{Local} = 6.2X + 5.3X$$
$$T_{Local} = 11.5X$$

and for the Grid case we have

$$T_{Grid} = T_{stage\text{-}dataset} + T_{stage\text{-}code} + T_{analyze}$$
$$T_{Grid} = T_{move} + T_{split} + T_{move\text{-}parts} + T_{stage\text{-}code} + T_{analyze}$$
$$T_{Grid} = 0.13X + 0.25X + (46 + 62/N) + 7 + 5.3X/N$$
$$T_{Grid} = 0.38X + 53 + (62 + 5.3X)/N$$

From these dependencies, we see two main conclusions. First, for large dataset (> ~10 MB), the time to transfer over the WAN dominates the process (6.2X vs. 0.34X) and it is much better to use the Grid. Secondly, for long analysis times, using the Grid gives you a 1/N decrease in the analysis time. Figure 5 shows the surfaces of time dependencies on dataset size and number of compute nodes for a sample case of Higgs Boson analysis on Linear Collider data. The Grid analysis (shown in blue), is clearly beneficial over the local equivalent (shown in gold), for large datasets and large number of compute nodes.

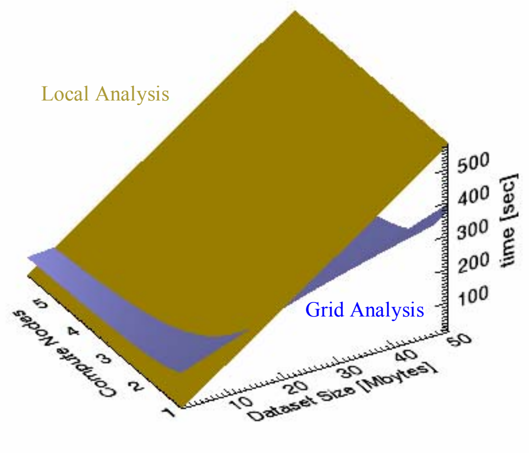


Figure 5. Analysis times (gold = local analysis, blue = Grid) as a function of dataset size and number of compute nodes.

5 Related Works

In an effort similar to this work, the Condor [15] project constructed the Distributed Batch Controller (DBC) framework [16] that processes scientific data over the Internet. The DBC system differs from our system at a fundamental level. The DBC stages data and distributes executables while IPA provides a development environment for scientists to dynamically and interactively construct code to analyze the data. In IPA, only a small amount of code needs to be re-distributed as the user customizes and rapidly develops the analysis code. In this sense, the IPA client is a full service integrated development environment for interactive Grid computing rather than batch computing.

There are as many attempts of comprehensive solutions for data analysis systems as there are batch job submission systems including all those associated with the Grid. Many work well within the design parameters meant for batch systems. However, what we are addressing in this paper is the more complicated case of interactive analysis where the user needs an agile and responsive environment.

6 Conclusion

We have described a framework and a reference implementation that allows users to do interactive analysis on Grids such as the Open Science Grid. The framework is not specific to any particular science application, although it does require record-based data. Our reference implementation is particularly suited for high-energy physics data analysis, but the framework can easily be adopted for applications in other fields, such as chemistry and biology, with a very high degree of reusability.

We have shown that our reference implementation has a performance that is suitable for interactive dataset analysis. We have also shown that the framework allows for a generically defined dataset that goes beyond sets of physical files and even logical file descriptions. The system is particularly advantageous over a local analysis on a single processor for large datasets and for complicated analysis algorithms.

Finally, we have shown that the main additional requirements of the Grid submission site for interactive analysis as we have defined it are the need for a fast processing queue and a path unblocked by a firewall from the computational nodes back to the client that is to receive the merged results. With these requirements satisfied, authors can use our framework to build interactive dataset analysis services for their Grids.

7 Acknowledgements

This project was supported by the D.O.E. under the SBIR Grant DE-FG03-02ER83556 and Tech-X Corporation. We are also grateful to Dr. David Smithe for helping with Figure 5.

8 References

[1] F. Berman, G.C. Fox, A.J.G. Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd., New Jersey, 2005. The Global Grid Forum: http://www.ggf.org. The Globus Alliance: http://www.globus.org.

[2] I. Foster, C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, 15(3), 2001.

[3] The Open Science Grid Project http://www.opensciencegrid.org/

[4] Enabling Grids for E-sciencE (EGEE) project http://public.eu-egee.org/

[5] The Large Hadron Collider Accelerator at CERN http://lhc.web.cern.ch/lhc/

[6] The International Linear Collider at SLAC http://home.slac.stanford.edu/aboutilc.html

[7] Globus Toolkit homepage: www.globus.org

[8] The W3C SOAP Specification. http://www.w3.org/TR/soap/

[9] The Java Analysis Studio http://jas.freehep.org/jas3/

[10] The WS-Resource Framework. K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe. March 5, 2004.

[11] Globus Resource Allocation Manager (GRAM), Globus Project, 2006, http://www-fp.globus.org/gram/overview.html

[12] The PNUTS project -- http://pnuts.org/

[13] ROOT - An Object Oriented Data Analysis Framework http://root.cern.ch/

[14] AIDA - Abstract Interfaces for Data Analysis http://aida.freehep.org/

[15] Condor – High Throughput Computing http://www.cs.wisc.edu/condor/

[16] C. Chen, K. Salem, and M. Livny, "The DBC: Processing Scientific Data Over the Internet", *16th International Conference on Distributed Computing Systems*, May 1996.