SERVOGrid Complexity Computational Environments (CCE) Integrated Performance Analysis

Galip Aydin, Mehmet S. Aktas, Geoffrey C. Fox ,Harshawardhan Gadgil,
Marlon Pierce and Ahmet Sayar
Community Grids Lab, Indiana University, Bloomington, Indiana, 47404, USA
{gaydin, maktas, gcf, hgadgil, mpierce, asayar}@cs.indiana.edu

Abstract

In this paper we describe the architecture and initial performance analysis results of the SERVOGrid Complexity Computational Environments (CCE). The CCE architecture is based on a lightly coupled, Service Oriented Architecture approach that is suitable for distributed applications that are tolerant of Internet latencies. CCE focuses on integrating diverse Web and Grid Services for coupling scientific applications to Geographical Information Systems. The services and coupling/orchestrating infrastructure are mapped to problems in geophysical data mining, pattern informatics, and multiscale geophysical simulation.

1. Introduction

We describe the initial testing of the integration and performance of several component pieces being developed as part of the SERVOGrid Complexity and Computational Environment (CCE). The overall CCE architecture is described in the technical report, “Complexity Computational Environment (CCE) Architecture,” [1]. To briefly summarize this document, the primary components of the system are as follows:

1. Earthquake simulation and modeling codes to support data assimilation, data mining, and multiscale modeling;
2. Data modeling with ontologies and semantic tools;
3. Web Services for managing data sources, data flow, code execution, and information;
4. A message-based event system (NaradaBrokering); and
5. A workflow management system (HPSearch) [11] for connecting services.

In this document, we explicitly focus on points 3, 4, and 5.

The goal of the CCE system is to build and integrate different domains of Grid and Web Services into a single cooperating system. Science Grids have tended to focus on code execution, job management, and high performance data transfer. However, projects such as LEAD [23] and GEON [24] have demonstrated the importance of integrating scientific computing with online meteorological and geophysical data sources. In our SERVOGrid work, the importance of both archival and real-time geophysical data has led us to implement a number of Web Services to support Geographical Information Systems. In this paper, we present an initial integration of these various services.

To establish some performance measurements on a particularly interesting application, we chose the Pattern Informatics (PI) application [14] as a test case. A screen shot of the user interface is shown in Figure 1. Our distributed architecture is an iteration of the earlier RDAHMM data mining tests documented in [2]. The current tests extend this earlier system by adding Geographical Information Services (GIS) -- the Web Map Service [21] and Web Feature Service [20] -- and the WS-Context [12] information service. We designed and developed all services described in this paper. More information and WSDL interfaces are available from [6, 7, 11, 12].

We chose the PI application since it is used to produce the well-publicized “hot spot” maps published by SERVO team member Prof. John Rundle and his group at the University of California-Davis. The integrated PI-GIS system profiled here, when in production, represents a very high-profile application of SERVOGrid, so the current tests provide an important practical example. The PI code is also simple to run, reliable, and fast, simplifying our tests.

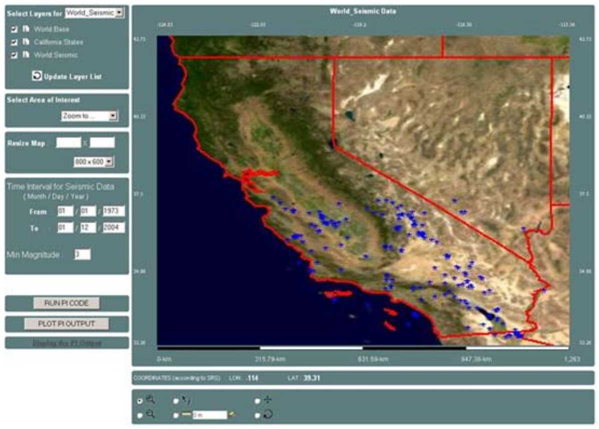


Figure 1. The SERVOGrid WMS client interface creates overlay maps that can also be used to set up and launch the Pattern Informatics code.

The PI case study provides more general insights into overall system performance because the GIS services provide access to GPS, seismic event, and fault catalogs, and so are important parts of many CCE applications such as GeoFEST, RDAHMM, Virtual California, and Potts Model codes. Also, the HPSearch workflow engine provides a general purpose service management system and transparently manages data flow between service components. Hence it is imperative to test the overhead introduced by this scheme.

This report documents our efforts welding the various pieces of SERVO together and has served to point out various bottlenecks in the current system, as described below. We report specific lessons learned for improving performance in applying conventionally designed Java Web Services to problems with non-trivial data requirements. We do not anticipate major changes in the architecture as the system evolves, but we do anticipate interesting Web Service performance research will be required to overcome some performance problems inherent to HTTP-based Web Services. In particular, we plan to address some of these issues using techniques described in [3] to improve the performance of the data services.

2. System Overview and Components

Web Feature Service: The Web Feature Service (WFS) [20] is an Open Geospatial Consortium (OGC) [4] based data service that manages "feature" data: abstract representations of map features and associated metadata. WFS servers are conventionally used by the GIS community to store map entities such as political boundaries and geographic features (rivers, roads), and so forth. We may also use the WFS to store information sets specific to the SERVO modeling code requirements, including a) GPS data archives, accumulated from SOPAC, JPL, and USGS data archives; b) seismicity data archives, accumulated from SCSN, SCEDC, Dinger-Shearer, and Haukkson formatted data archives; and c) earthquake faults obtained from the QuakeTables fault database [5].

We have designed and built a Web Services-based WFS [6], which we use in these tests. The WFS stores data accumulated from public, online archives. We reformat and store locally, to support query searches and combined/filtered results. More information is available from [10].

Web Map Service: The Web Map Service (WMS) [21] is an OGC specification for generating interactive, online maps. WMS can generate maps in several formats (JPEG, SVG) by acting as client to both WFS and other WMS instances. WMS maps are generated as overlays, so it is possible to generate and customize maps interactively. The SERVOGrid WMS [7] interacts with the NASA OnEarth WMS [8], which provides very high quality satellite images. WMS implementations can also be used to interactively extract component features. We use the SERVOGrid WMS to, for example, set up initial problems visually with information from the WFS (such as seismic event records described below or earthquake faults), then "read off" the WFS from the map to generate input files for running SERVOGrid earthquake modeling codes. For a general overview of WMS and related material, see [9].

We designed and developed a Web Services-compatible Web Map Service for this project. All performance measurements are made using this implementation. More information on the SERVOGrid WMS implementations can be found at [10].

HPSearch: The HPSearch [11] system provides a scripting environment for managing distributed Web Services. We specifically use HPSearch for deploying system components and for managing the distributed services in a workflow pattern on SERVOGrid. For the current system, we rely on a HPSearch node, which is responsible for managing the execution of services, and WSProxyService [17], a specialized Web Service that functions as both, a Web Service and a NaradaBrokering publisher and/or subscriber.

Context Service: The Context Service [12] is a system for storing transitory metadata needed to describe distributed session state information. In the current test system, it is used to store information needed by HPSearch to orchestrate system interactions.

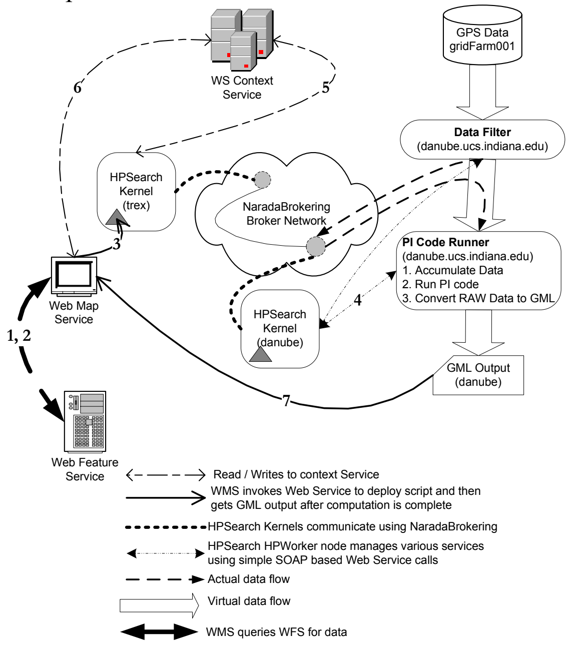
NaradaBrokering: NaradaBrokering [13] development is not directly part of the current work, but we rely upon it to manage data flows and event notifications within the SERVOGrid system. NaradaBrokering may be thought of a as topic-based publish/subscribe messaging system: interested entities can register to a NaradaBrokering node to send and receive messages on particular topics. HPSearch uses NaradaBrokering to route data streams.

3. Performance Tests

Computers and Networks: All tests were run at the Community Grids Lab using modestly performing PCs and Linux servers. CGL computing environment consists of Windows (2000 and XP), Linux, Sun Solaris workstations and servers, including 40 Pentium 4-based desktop class machines, 20 Linux/Solaris (dual-CPU) server class, and two 8-CPU 16 GB Sun v880 server class machines to support the lab's development and research efforts. The laboratory's network consists of server, workstation, and mobile connectivity provided by 100Mbit/second Ethernet, and 11Mbit/second 802.11b wireless connections respectively, connected in turn to Indiana University's network backbone via a 1000Mbit/s fiber optic link, and from there to the Internet2 Abilene network and the commodity internet via multiple OC3 links.

We run the WFS, WMS, PI code, and Context Service each on separate servers, as described in "Test Scenario." The purpose of these tests is to establish relative performance numbers and identify bottlenecks, not to establish absolute times or benchmarks.

Running the Test Application: The Pattern Informatics (PI) code is used to generate "hot spot" maps based on past seismic records: it identifies areas of high probability for large future earthquakes. The PI code is described in more detail in [14]. For our purposes, we only need to be concerned with the requirements for running the code. It needs these inputs: a) a seismic record catalog (SCEDC) [15] file with dates, latitude/longitude coordinates, and event magnitudes; b) time and space boundaries for selecting the region and time period of interest from the seismic catalog; and c) a lower bound for seismic event magnitudes: seismic events below the desired threshold are discarded. Note this threshold dramatically affects the size of the input data file since the number of events grows exponentially as the lower bound decreases (see Table 1). PI generates an output file consisting of a latitude/longitude grid of probabilities (forecasts) for events within a given future time period.



WS Context Service
GPS Data gridFarm001
6
5
Data Filter (danube.ucs.indiana.edu)
HPSearch Kernel (trex)
NaradaBrokering Broker Network
3
PI Code Runner (danube.ucs.indiana.edu)
1. Accumulate Data
2. Run PI code
3. Convert RAW Data to GML
Web Map Service
4
HPSearch Kernel (danube)
1, 2
GML Output (danube)
7
Web Feature Service
Read / Writes to context Service
WMS invokes Web Service to deploy script and then gets GML output after computation is complete
HPSearch Kernels communicate using NaradaBrokering
HPSearch HPWorker node manages various services using simple SOAP based Web Service calls
Actual data flow
Virtual data flow
WMS queries WFS for data

Figure 2. Architectural diagram of the integrated system. Host names are given in parenthesis.

The PI application is a natural candidate for testing the SERVOGrid WFS, WMS, HPSearch and Context services because the local copy of the SCEDC catalog file can be replaced by the SERVOGrid WFS, and the WMS can be used to interactively set up the problem, allowing a user to visually set latitude/longitude bounding boxes and time intervals. After the problem parameters are configured through WMS interface, HPSearch can be used to execute the simple workflow associated with the PI code: extract data from the WFS, transfer it to the PI code's host, execute the PI code, and notify listeners when the output data is available. Finally the WMS can be used to visualize the hotspot outputs. In this scenario the Context Service can be used to store various pieces of information, such as the state of the system ("executing" or "done") and the location of data files.

Following steps summarizes how system works (see Figure 2. We will expand on this in more detail in the "HPSearch Performance Tests" subsection.

1: WMS queries WFS for a given bounding box and time interval

2: WFS dumps the results into a web accessible ASCII file (see comment below).

3: WMS starts a session, invokes HPSearch to run workflow script for PI Code with a session id

4: HPSearch runs the workflow script and generates output file in GML format as result
5: HPSearch writes the URI of the of the output file into Context
6: WMS polls the information from Context Service
7: WMS retrieves the generated output file's location from the Context Service, downloads the data, and generates a map.

Note that Step 2 is a temporary solution. Normally WFS returns the query results as a GML Feature Collection, but since we did not want to impose GML processing overhead on HPSearch, we devised this intermediary step where WFS dumps the query results to a file for HPSearch before converting them to GML to transfer to WMS. We are developing streaming support both for WFS and WMS which will allow us to remove Step 2.

We calculated the performance of the various system components for the following lower bounds for the seismic event magnitudes: M=5.0, 4.5, 4.0, 3.5, and 3.0. These correspond to increasing data file size, as shown in Table 1 for the eventful year 1992 in the SCEDC catalog for Southern California. The entire catalog from 1932 to 2004 has 401,403 entries.

Table 1. SCEDC entries for test year 1992.

Event Magnitude Lower Bound	Number of Seismic Events	GML Result Size (KB)
5.0	19	11
4.5	67	36
4.0	209	106
3.5	587	287
3.0	1790	880

WFS Performance: We measure WFS performance by timing the steps needed to extract seismic records with specific latitude/longitude bounding boxes, time periods, and lower bounds for the earthquake threshold magnitudes. These extracted records are returned as GML responses. This test is representative of other SERVOGrid applications that need to extract records from remote data bases through the WFS. The tests are made over 10 runs. Data from 1/1/1992 to 12/31/1992 were requested and latitude/longitude bounding box (32.0, -117.0)-(37.0, -114.0) was used.

We make 4 types of measurements in addition to the total processing time between receiving the getFeature request and returning the feature collection object: Initialization time is spent during object initializations and checking to see if the database that contains the requested feature is alive. This is a relatively small period of time (average 30ms) and can be ignored. During the initialization phase, the WFS extracts the query from the request and opens a connection to database that has the requested feature data. We measure the total query execution time since it affects the performance significantly. Other important measurements are made to find out how much time it takes to build GML feature objects from the query results and then how long it takes for merging them into a feature collection as the final result to return.

WFS test results (Figure 3) show that the performance decreases as the lower event threshold M decreases, corresponding to the increase in data size. Higher thresholds are dominated by database query execution time which remains relatively constant for all magnitudes. However for event thresholds 3.5 and 3 time for building GML object takes considerable amount of time. Our further tests showed that for event magnitudes between 2.5 and 1 this step dominates the total processing time.

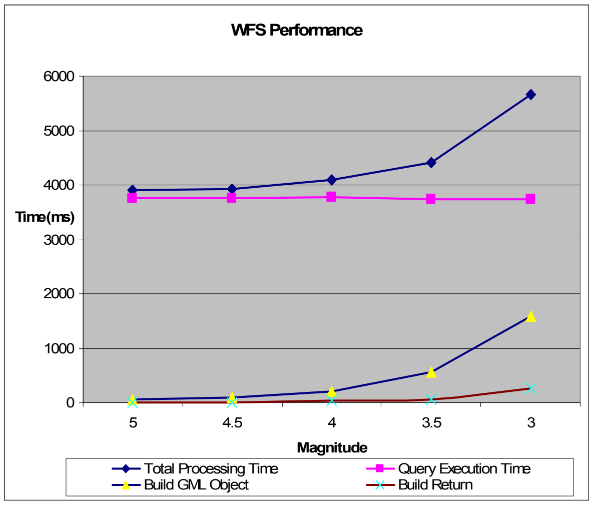


Figure 3. WFS performance measurements

We note that the results depicted in Figure 3 are intermediate results which show significant improvements over initial testing results. Although the performance is almost the same for lower event threshold M=5 and 4.5 it is improved by a factor of 4.4 for M=4, 4.7 for M=3.5 and 22.8 for M=3. Since our initial design was intended for smaller data loads GML FeatureCollection objects were created using string concatenation methods. But this proved to be highly costly for larger data sizes and we utilized temporary files for immediately flushing the intermediate results and using byte arrays instead of strings for creating the final results. This improved performance significantly for M > 4. Currently the most significant portion of the total processing time is spent for database query execution and we are investigating methods to optimize it. The initial WFS performance analysis results are available in an earlier version of this report [26].

The current version of our WFS implementation is based on interacting relational databases such as MySQL, but we are planning to test support for native XML databases as well. The latest version of SERVO Grid WFS binary and source files as well as the related schemas are available from WFS web site [6].

HPSearch Performance: The HPSearch engine controls the flow of data from the WFS to the data filtering service that filters and reformats the data and then to the PI service that manages the execution of PI code. As the system goes through these stages, HPSearch interacts with the Context Service to store metadata and keep track of state. There are two HPSearch nodes in the system: a master node, and a worker node. The master node is responsible for scheduling worker HPSearch nodes: there is only one worker in the demo, but for load balancing and scheduling we can run additional worker nodes.

In addition to the WFS and Context Service HPSearch nodes interact with a Data Filter service and the PI Code Runner service. The *Data Filter* service is responsible for translating the GML output of the WFS into a format understood by the PI code, and for transmitting this reformatted data to the PI Code Runner. The *PI Code Runner* is also an extension of the WSProxyService class and consists of three related services: a Data Accumulator, which receives data from the Data Filter; a code wrapper, which can execute the PI code; and a "Raw-to-GML" translator, which translates the PI code output to GML, for later display in the client as a hotspot map. These services are co-located in our tests but may be separated.

HPSearch uses NaradaBrokering nodes to route data streams between components and also for exchanging control messages between different HPSearch nodes.

The system works as follows (Figure 2). We denote the HPSearch master node as HPMaster (runs on the host *trex*), which also acts as a Web Service, and the worker node as HPWorker (runs on the host *danube*).

The end user selects a bounding box, time interval and lower seismic event threshold for the problem. In the user interface we also provide specific UI elements for deploying the flow. The deployment is started by clicking the "Run PI code" button which invokes the Web Service with the appropriate parameters, on the HPMaster node submitting the flow for execution.

The script which defines the flow structure initially creates a placeholder entry in the context service and marks the status of the flow as "Executing". The script further sets up the various components of the flow (components wrapped as Web Services using the WSProxy) and initializes them. Initialization entails setting up the various communication channels thus linking the individual components via streams using NaradaBrokering.

After a successful initialization, the HPMaster node signals the Data Filter component to start the flow. The Data Filter components downloads the input data, filters and reformats each line of the data and streams it to the PI Code Runner service. After the entire data is collected, the PI Code Runner service executes the PI code on the data followed by conversion of the output data to GML format for further use. This marks the end of the flow and the HPSearch master node is notified of the successful flow completion. The HPMaster node then modifies the placeholder entry in the context service created previously, to reflect the URL of the final output GML file.

The WMS, in the meantime, continuously polls the Context service, checking to see if the service was completed. Once the HPMaster has updated the placeholder entry, the WMS can download the resulting GML file to plot the results.

The total HPSearch processing time contains a) the time required to create a placeholder entry in the context service, b) executing the flow and c) updating the placeholder entry. In addition to this we measure Data Processing time and the PI Code Runner Service time. Data Processing time is spent for reading, transporting, and re-formatting the GML output from the WFS into the legacy format expected by the PI code. The data is read line by line, transformed, and transported from the WFS to the PI Code Runner service. For the sample data this takes, on average, a little over eight seconds. The PI Code Runner Service consists of several co-located services. The first, *Data Accumulate*, receives data from the Data Filter service and writes it to a local file (required by the PI code executable). The *Exec PI Code* service then runs the PI application. For the test data, the PI code execution time is negligible. *Raw->GML* is a filter for transforming the PI output file into a GML format that can be interpreted by the Web Map Service.

The HPSearch overhead may be found by comparing the "Data Processing" time to the "Exec PI Code" time. Note that the Data Filter and Data Accumulate steps run concurrently. The overhead percentage is

$$Overhead = 1 - \frac{(Data\_Filter + Exec\_PI\_Code + Raw\_to\_GML)}{Elapsed\_Time\_For\_Flow}$$

The overhead percentage we calculated is about 4% for all values of M.

Note that from Figure 3 below, it takes about 16 – 18 seconds to run the PI code using the HPSearch system. Also note that the time required while running the PI code for different magnitudes is approximately the same. Our current architecture contains a data filter component which is responsible for filtering and reformatting the input data file to suit the PI code. This step reads all the input data (about 50000 lines) and filters each line with the given specifications. Since the data filtering step takes the maximum time, the overall time is dominated by this step and is constant for each run. Ref [22] contains more detailed overhead results for the same test. During the development of our initial prototype, we could improve the performance by a factor of 1000 by making the following modifications.

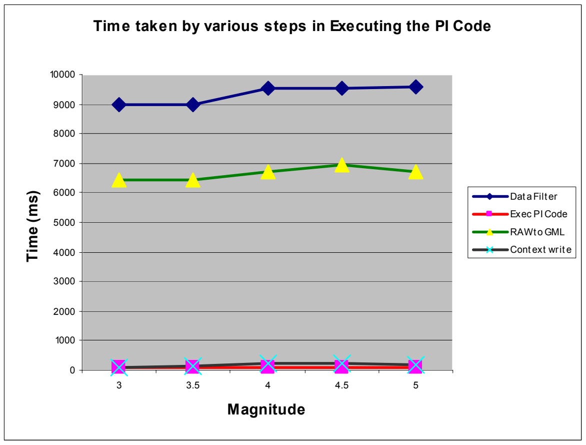


Figure 4. HPSearch performance measurements

First, HPSearch previously used the RunnableProxyService interface for hosting the data filter service. This data service checks after each run, whether the service STOP / PAUSE was requested. This extra step introduces a heavy penalty and we were able to reduce this penalty by a factor of about 50 by implementing the WrapperProxyService interface.

Second, the RAW data to GML format conversion service performance was also improved by introducing immediate flushing out of the relevant output streams rather than buffering the data, thereby improving the performance by a factor of about 20.

We plan to make future amendments to the prototype to further improve the performance. Currently the data accumulation step works in 2 parts, namely, the WFS creates a temporary file for string the query results, which is then filtered by the data filter service. By having HPSearch directly invoke the WFS and have it stream data to the PI code runner, we can remove the extra overhead of the data filtering and temporary file creation step.

Context Service Performance: The Context Service stores metadata that (collectively) identifies the state of the system, as shown in Figure 2. The metadata pieces are typically small and are independent of the event magnitude M used in the PI testing.

The Context Service's primary operations are GetContext and SetContext. Three measurement sets were made using a 50 byte string for GetContext. Each of the three sets consisted of 100 individual measurements. We also performed 3 sets of 100 measurements on the SetContext method. In average, we measure ~116 ms for GetContext and ~125 ms for SetContext functions to be performed. Both of these measurements are internal timings to process requests.

We conclude from this that the Context Service does not add excessive overhead to the overall system for these small metadata stores. The actual internal processing time for small metadata pieces is typically smaller than the network invocation time.

WMS Performance: The WMS generates maps based on input (bounding boxes and time intervals) from the user interface. Maps are generated in layers from features sets obtained from one or more WFS sources. WMS can also combine the locally generated images of features with images obtained from other (remote) WMS instances. In the current test case, our WMS generates map boundaries, locations of earthquake events, and locations of hotspots, which are superimposed on images obtained from the OnEarth WMS at JPL [8].

We tested WMS service performance for M=5.0, 4.5, 4.0, 3.5, and 3.0. We chose a latitude/longitude bounding box of (-124.85, 32.26), (-113.56, 42.75) and tested results for the time period 1/1/1992 to 12/31/1992 (the most active year in the catalog). The timings represent averages of ten measurements per point.

Our WMS Client (shown in Figure 1) is a thin client to the WMS for displaying the maps returned by the server. The WMS generates the images and delivers back to the client. The WMS operation timings are thus split between the internal server timings and the overall timings seen by the client. Note that the "internal" server timings, Figure 5, include remote calls to the WFS and OnEarth WMS. In addition to the initialization and total processing time we measured the times to retrieve the LandSat images for the selected latitude/longitude bounding box from OnEarth WMS and GML formatted seismic records from WFS. We also measure the time to assemble the abstract map pieces to render as a JPEG image.

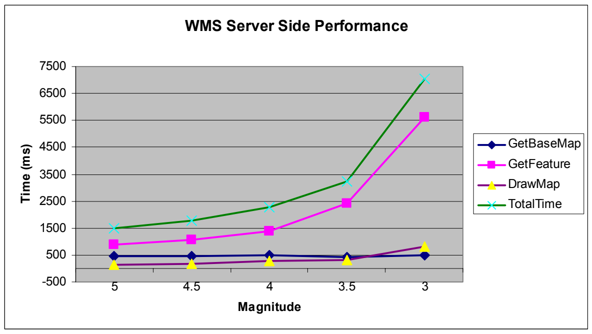


Figure 5. WMS server side performance

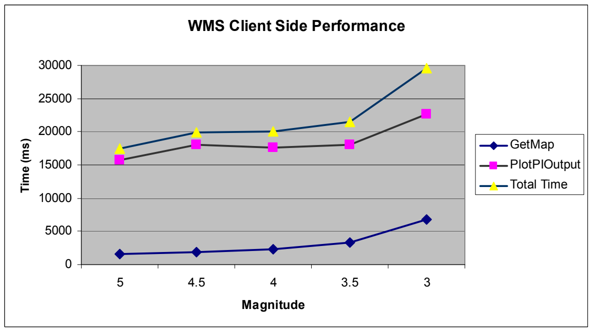


Figure 6. WMS client side performance

The WMS client side performance results (Figure 6) show that plotting the GML result dominates the total processing time.

4. Conclusions and Future Work

This initial performance evaluation and related tests have served to prove several of the basic concepts of CCE architecture, while revealing bottlenecks and areas of needed performance improvement.

We employ Web Service-based WFS to bridge data sources with client applications by using standard OGC interfaces. However, the HTTP protocol-based approach has its limitations: transfer of large volumes of data may cause long delay or may not be possible at all. To overcome this difficulty we are initially implementing a streaming version of WFS using NaradaBrokering as transfer medium [16]. There are also interesting techniques that use the XML Infoset to preserve the XML message in binary transmissions [25]. These techniques have been applied to hand-held devices but could apply equally well to large data transmissions from standard servers.

The WMS performance depends on WFS, HPSearch, and network speed. However, it also can be made more efficient. We plan to increase performance of both WMS server and client modules by employing new algorithms and optimization techniques such as using distributed rendering and tiling, and parallel rendering of images. Additionally future versions of WMS will be used for scientific visualization which requires us to handle high volumes of data. To support high performance large volume data transfer we are integrating our WMS implementation with NaradaBrokering.

We have demonstrated the use of HPSearch [2] to deploy and manage system components. However, currently the system can only handle one user at a given time. We are currently adding support for session management within HPSearch so that multiple users can execute the same flow while HPSearch transparently manages temporary data files created during execution of each instance. Further, HPSearch currently does not address security for data streams. We plan to leverage NaradaBrokering's security features [18] for securing data streams in the future.

5. Acknowledgements

We would like to thank James Holliday of the University of California-Davis for providing the PI code and data sets. Dr. Jay Parker at NASA JPL critiqued earlier versions of this document. Dr. Shrideep Pallickara provided assistance with NaradaBrokering. This work is supported by the Advanced Information Systems Technology Program of NASA's Earth-Sun System Technology Office.

6. References

[1] Geoffrey Fox, et al, "Complexity Computational Environments (CCE) Architecture." Technical report available from http://grids.ucs.indiana.edu/ptliupages/publications/CCE%20Architecture.doc

[2] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Robert Granat "A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications" Proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference (CCGrid 2005). Cardiff, UK May 2005

[3] Geoffrey Fox, Harshawardhan Gadgil, Shrideep Pallickara, Marlon Pierce, Robert L. Grossman, Yunhong Gu, David Hanley, Xinwei Hong, Indiana University and University of Illinois at Chicago "High Performance Data Streaming in Service Architecture" Technical Report July 2004

[4] The Open Geospatial Consortium, Inc. (OGC) Web Site: http://www.opengeospatial.org/

[5] Chen, A., Donnellan, A., McLeod, D., Fox, G., Parker, J., Rundle, J., Grant, L., Pierce, M., Gould, M., Chung, S., and Gao, S., "Interoperability and Semantics for Heterogeneous Earthquake Science Data", International Workshop on

Semantic Web Technologies for Searching and Retrieving Scientific Data, Sanibel Island, FL, October 2003.

[6] SERVOGrid WFS implementation web page: http://www.crisisgrid.org/html/wfs.html

[7] SERVOGrid WMS implementation home page: http://complexity.ucs.indiana.edu/~asayar/wms

[8] OnEarth: Server of Landsat 7, WMS Global Mosaic. Web Site: http://wmt.jpl.nasa.gov/.

[9] Ahmet Sayar, Marlon Pierce, Geoffrey Fox "OGC Compatible Geographical Information Services", Technical Report (Mar 2005), Indiana Computer Science Report TR610. Available from http://grids.ucs.indiana.edu/ptliupages/publications/ogctech_report.pdf

[10] Crisis Grid Web Site, Geographic Information Systems Research at Community Grids Lab, http://www.crisisgrid.org

[11] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara "HPSearch for Managing Distributed Services", Work in Progress session at IEEE/ACM Cluster Computing and Grid 2005 Conference (CCGrid 2005). Cardiff, UK May 2005. HPSearch Web Site, http://www.hpsearch.org

[12] Mehmet S. Aktas, Geoffrey C. Fox and Marlon Pierce, "Managing Dynamic Metadata as Context, Technical Report", available at: http://grids.ucs.indiana.edu/ptliupages/publications/maktas_iccse05.pdf. See also Context Service web site: http://grids.ucs.indiana.edu/~maktas/fthpis

[13] Shrideep Pallickara and Geoffrey Fox "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids", in Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003, Rio Janeiro, Brazil June 2003. See also the NaradaBrokering Web Site: http://www.naradabrokering.org

[14] Tiampo, K. F., Rundle, J. B., McGinnis, S. A., & Klein, W., "Pattern dynamics and forecast methods in seismically active regions", Pure Ap. Geophys. 159, 2429-2467 (2002).

[15] Southern California Earthquake Data Center (SCEDC) Web page, http://www.data.scec.org/

[16] Galip Aydin, Marlon Pierce, Geoffrey Fox, "High Performance Web Feature Service Implementation for GIS Grid and Web Service Architectures", Submitted to GML And Geo-Spatial Web Services Conference 2005.

[17] Harshawardhan Gadgil, Jin-Yong Choi, Bernie Engel, Geoffrey Fox, Sunghoon Ko, Shrideep Pallickara, Marlon Pierce, *"Management of Data Streams for a Real Time Flood Simulation"*,, CGL Technical Report June 2004, Available from http://grids.ucs.indiana.edu/ptliupages/publications/crisisGridFlow.pdf

[18] Yan Yan, et. al., "Implementing a Prototype of the Security Framework for Distributed Brokering Systems". Proceedings of the 2003 International Conference on Security and Management Volume I pp 212-218

[19] Shrideep Pallickara, Geoffrey Fox and Hasan Bulut "Fault Tolerant Reliable Delivery of Events in Distributed Middleware Systems", Technical Report April 2005. Available from http://grids.ucs.indiana.edu/ptliupages/publications/FT-ReliableDelivery.pdf

[20] Vretanos, P. (ed.), "Web Feature Service Implementation Specification", (WFS) 1.0.0, OGC Document #02-058, September 2003.

[21] Jeff De La Beaujardiere, OpenGIS Consortium "Web Mapping Server Implementation Specification", 1.3, OGC Document #04-024, August 2002.

[22] More information about the PI demo is available at: http://www.hpsearch.org/demos/PI

[23] Beth Plale, Dennis Gannon, Dan Reed, Sara Graves, Kelvin Droegemeier, Bob Wilhelmson, Mohan Ramamurthy, "Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD", *To appear ICCS workshop on Dynamic Data Driven Applications*, Atlanta, Georgia, May 2005. See also http://lead.ou.edu.

[24] GEON (Geosciences Network): A Research Project to Create Cyberinfrastructure for the Geosciences. http://www.geongrid.org.

[25] Sangyoon Oh, Hasan Bulut, Ahmet Uyar, Wenjun Wu, Geoffrey Fox, "Optimized Communication using the SOAP Infoset For Mobile Multimedia Collaboration Applications", Proceedings of the International Symposium on Collaborative Technologies and Systems. May 2005, Missouri, USA.

[26] SERVOGrid Initial Performance Analysis, CGL Technical Report, available from: http://www.crisisgrid.org/html/servo.html