High Performance Data Streaming in Service Architecture

Geoffrey Fox[†] Harshawardhan Gadgil[†] Shrideep Pallickara[†] Marlon Pierce[†]
Robert L. Grossman[‡] Yunhong Gu[‡] David Hanley[‡]
Xinwei Hong[‡]
[†]Community Grids Lab, Indiana University, IN, USA
501, N. Morton St. Suite 224, Bloomington, IN - 47404
hgadgil@cs.indiana.edu, {gcf,spallick,marpierc}@indiana.edu
[‡]Laboratory for Advanced Computing, University of Illinois at Chicago
700 SEO, MC249, 851 S. Morgan St., Chicago, IL 60607, USA
grossman@uic.edu, {gu, dave, xwhong}@lac.uic.edu

Abstract

Applications dealing with large data sets obtained via simulation or actual real-time sensor networks are increasing in abundance. The data obtained from real-time sources may contain certain discrepancies which arise from the dynamic nature of the source. Furthermore, certain computations may not require all the data and hence this data must be filtered before it can be processed. By installing adaptive filters that can be controlled in real-time, we can filter out only the relevant parts of the data thereby improving the overall computation speed. In this paper we present an architecture that links these distributed filters to achieve high throughput dataflow for real-time streaming and high-performance applications.

Keywords: High Performance Computing, Stream Data Processing, Crisis Management, Data Mining, Assimilation and Analysis, Geographic Information Systems

1 Introduction

We believe that over the next few years, there will be growing interest in extending use of high end computing platforms from simulation to the data mining and data analysis of (multiple) high volume data streams. We expect data analysis, data mining, and data assimilation to grow in sophistication and to merge into an infrastructure that can not only analyze data at rest, but also to integrate and to mine, in real time, very high volume data streams. We describe prototype software tools for exploring, mining, filtering and assimilating streaming data distributed over extreme scale computing platforms consisting of thousands of parallel and distributed nodes.

High-end computing applications increasingly require the real-time integration of streaming data with high-end computation. One class of examples includes data mining multiple streams of real time data or data streams from multiple repositories. Another class of examples includes scientific data analysis and assimilation, including the analysis and assimilation of earth and environmental science data (as in earthquake, flood and weather forecasting), astronomical data (as in virtual observatories), bioinformatics data (as in drug discovery), and critical infrastructure simulations.

In the following section we give one example for each class - *Intelligence Data Analysis* and *Earthquake and Flood Grid Data Assimilations* - on which we are testing our tools and prototype data handling and streaming technology. We then illustrate how we can use NaradaBrokering (Section 4.1) to alleviate issues related to fault tolerance and reliable delivery of real time streaming data for high performance computing applications. Further we also propose an architecture to achieve high throughput data transfer between components (data filtering applications) with minimum overhead.

The rest of the paper is organized as follows. In Section 2 we present the motivation for an architecture for *Extreme Scale Computing* and illustrate the same with two applications - *Crisis Grid* and *Distributed Collaboration System*. Section 3 introduces the architecture details. Section 4 introduces the tools - *NaradaBrokering*, *HPSearch* and *WSProxy* - we use, in realizing this architecture. Section 5 proposes an architecture to achieve high throughput in streaming data transfer and presents a few results with an initial prototype. Section 6 gives an overview of the future work. Section 6 is conclusion.

2 Motivation for Extreme Scale computing

2.1 Intelligence Data Analysis

The term *Intelligence Data Analysis* stems from the process of efficiently processing very large amounts of real time data, extracting relevant information, formulating appropriate actions semi-automatically and getting the right recommendation to the right decision maker at the right time. *Situational Awareness* describes an integrated view of elements and threats impacting a single location. Computing situational awareness in real time requires the processing and integration of many real time data streams, including the analysis of real time streams, such as image data, radio frequency data, signals intelligence data, communications data, open source streaming data and data streams from more specialized collection devices. In traditional *"Intelligence Data processing"* we collect, process and warehouse the data, generally by source and type of data. This warehoused data is analyzed by intelligence analysts and reports are produced which are then approved, integrated and disseminated. With actionable intelligence, we can supplement the process with real time process as follows

- Consumers access data streams in real time, process them as per their needs, and warehouse the streams as required.
- Multiple data streams are integrated in *real time*, enriched with data from databases and repositories and processes to create integrated summaries which provide situational awareness as required by consumers.
- This also helps in taking actions in real time such as
  - Focusing the attention of a decision maker (*Focus*)
  - Collecting more data (*Tipping and Cuing*)
  - Issuing alerts (*Indicators or Warnings*)

2.2 Earthquake Science Grid

The integration of real time streaming data and simulation is of growing importance in many areas that benefit from the increasing deployment of sensors, such as earth and environmental science. Earthquake data assimilation and data mining models such as *Hidden Markov Methods* rely on access to seismic catalogs and GPS station data sources. Typically these catalogs are available online as monolithic downloads that are filtered by hand editing. These are useful for archival purposes and for code development / validation but not suitable for real time data processing. With initiatives such as SCIGN (Refer to `http://www.scign.org`) we have access to GPS data using a Web Service interface enabling both user-based and application-based access to remote data. The advantage of integrating data streams in real time, enriched with data from data repositories allows us to take actions in real time, such as that desired by crisis response communities.

We have identified a couple of real-time applications where real-time data stream processing is useful.

2.3 Crisis Grid

Crisis Grid is a Grid based modeling framework that proposes to connect data-sources, high-performance modeling applications, computing resources, visualization services and collaboration services. High quality, high resolution data sources such as LIDAR (LIght Detection And Ranging) and NEXRAD (NEXt generation RADar) are available at unprecedented levels and the addition of small, networked sensor devices continue to drive the trend. With more and more data sources becoming accessible as Web Services, these services may be coupled with various core Grid Services to create a crisis response system for emergency preparedness, response and disaster informatics communities. This includes weather forecasting, earthquake modeling and flood modeling among other systems. Crisis Grid's infrastructure is constructed from core Grid services combined with special set of auxiliary Grids. The challenge lies in creating a unifying framework that connects these services in standard ways.

Ref [1] illustrates how we can build a *streaming data* based application to construct a flood modeling application involving distributed services. In the near future, we can look forward to a closer coupling of high performance simulation applications to real time data sources for near real time forecasting.

2.4 GlobalMMCS

The GlobalMMCS (Global Multi Media Collaboration System) [2] project tries to build a collaboration system which integrates various services including videoconferencing, instant messaging and streaming and support multiple videoconferencing systems and heterogeneous collaboration environments. Ref [3] describes a Web Service based streaming gateway that enables disparate clients to join in a collaborative environment and share real-time audio and video data. This process involves passing the data through various filters such as noise cancellation, silence detection, audio format conversion, audio mixer and video stream decoders and encoders among others. Since operations related to video mixing are computationally intensive, we may consider off-loading some parts of the processing such as format conversion, to distributed filters (Ref Section 3.2) that process streaming data. This filtered stream may then be sent to the compute cluster to generate the composite audio / video stream.

3 An Architecture for High Performance Data Streaming

3.1 Architecture

We consider the particular case where a set of data sources (sensors of repositories) are feeding data to a central HPC (High Performance Computing) system as shown in Figure 1. Each sensor is augmented with a data filter that controls the type and amount of data sent to the central HPC system. Each of these filters is also available as a Web Service, thus allowing us to dynamically discover and invoke them. Furthermore, the filters may pass data to other filters before the data is sent to the HPC system for processing, thus establishing a chain of filters connected using pipes. The HPC system is also available as a Web / Grid Service and hence a particular process may be invoked to do the required processing.

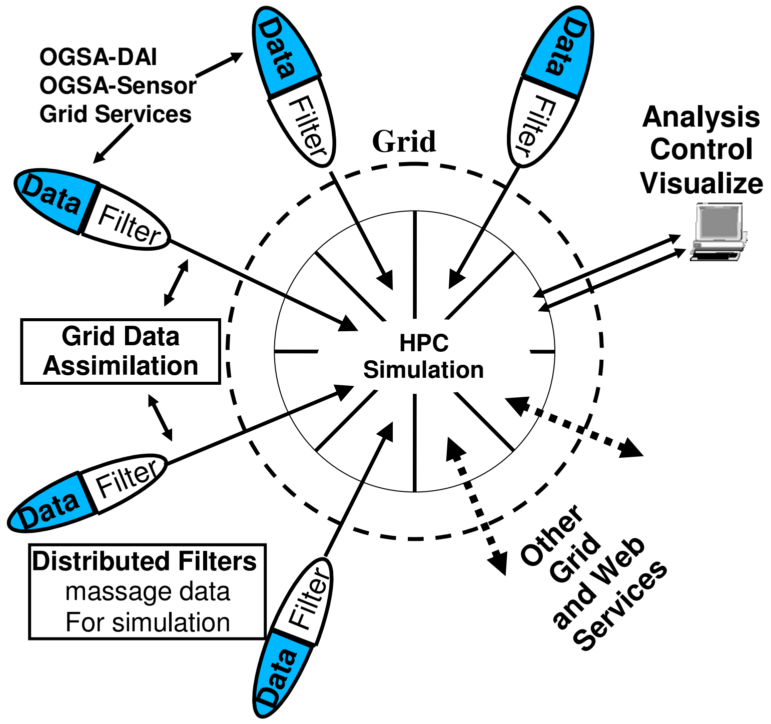

Figure 1: Data Deluged Computing Architecture

3.2 Data Assimilation

Due to data deluge in applications, the number of observed data points is much larger than the number of points required for the computation. In order to gain more efficiency in data processing, we install filters which are applications that operate on raw data and send out only the most important parts of the data. Filtering techniques such as *Principal Component Analysis* may be applied to remove insignificant data from the observed data set thereby reducing the cost of computation. As an illustration, consider the solving of some optimization problem involving a Kalman Filter like structure

$$\min_{TheoreticalUnknowns} \sum_{i=0}^{N_{obs}} [Data_i(position, time) - Simulated\_value]^2 / Error_i^2$$

Natural approach is to filter out each (position, time) pair so that only the important data combinations are obtained thereby relieving the HPC cluster of large error or insensitive data. An advantage of this scheme is that the computing cluster does not need to filter or process undesired data, thus resulting in a faster computation. This kind of filter based approach is also useful for real-time streaming data sources where, undesired data might creep into the unprocessed data and needs to be handled by the data processing application. For example garbled frames in a real-time video application or incorrect readings due to damaged sensors in a sensor network.

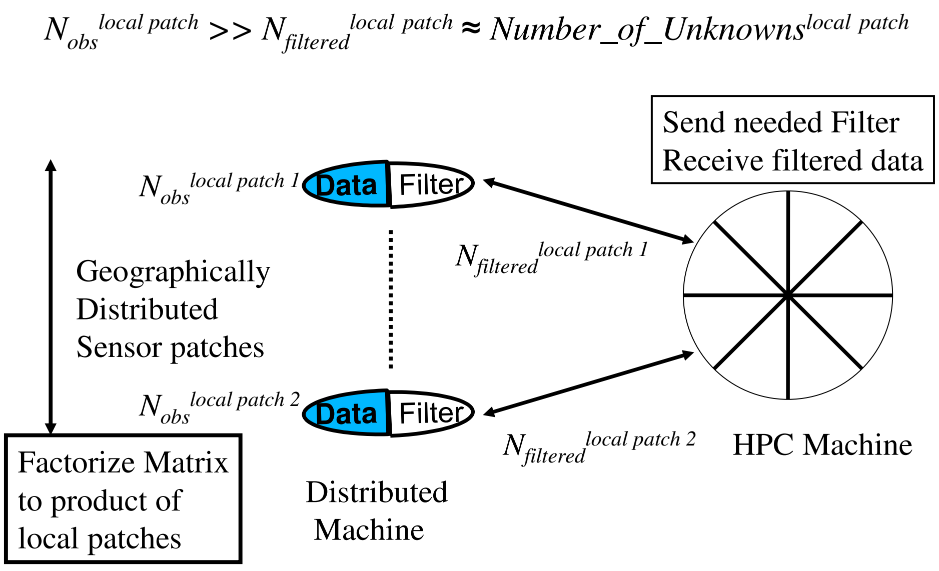
$N_{obs}^{local\ patch} >> N_{filtered}^{local\ patch} \approx Number\_of\_Unknowns^{local\ patch}$
Send needed Filter
Receive filtered data
$N_{obs}^{local\ patch\ 1}$
Data
Filter
$N_{filtered}^{local\ patch\ 1}$
Geographically Distributed Sensor patches
$N_{obs}^{local\ patch\ 2}$
Data
Filter
$N_{filtered}^{local\ patch\ 2}$
HPC Machine
Factorize Matrix to product of local patches
Distributed Machine

Figure 2: Distributed Filtering Architecture

Further, the data needed by a simulation depends not only on the data but also on the simulation. Simulations often dynamically affect the down stream requirements of the application. For this reason, our architecture assumes a feedback model where the distributed filters are controlled by the simulation. Control feedback may also be used to control and synchronize multiple distributed data streams to optimize the overall processing of data. Figure 2 illustrates how the teraflow data mining services architecture adaptively filters teraflow data flows using techniques such as linear transformations corresponding to singular value decomposition of least squares matrix.

In traditional control systems (using GridAnt [4] for instance), the generated data is accumulated in a central repository and then this data is transferred to the compute cluster using protocols such as GridFTP [5]. During processing the compute cluster may generate more data and this data must be stored until the computing is finished. This process requires a huge amount of data storage for temporary data. Thus processing data in a stream is advantageous since the data processing can begin on the fly thereby alleviating the need for extra temporary storage for gathered data.

4 Building blocks

In this section we introduce NaradaBrokering - a distributed routing substrate, HPSearch - a scripting interface to orchestrate services and control NaradaBrokering and WSProxy - a Web Service based proxy that allows us to process streaming data.

4.1 NaradaBrokering

NaradaBrokering [6, 7] is an event brokering system designed to run on a large network of cooperating broker nodes. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. NaradaBrokering guarantees delivery of events in the presence of failures and prolonged client disconnects, and ensures fast dissemination of events within the system. Events could be used to encapsulate information pertaining to transactions, data interchange, system conditions and finally the search, discovery and subsequent sharing of resources. We may summarize some of the important features of NaradaBrokering as follows

- Implements high-performance protocols (message transit time less than 1 ms per broker)
- Order-preserving optimized message transport
- Quality of Service (QoS) and security profiles for sent and received messages
- Interface with reliable storage for persistent events, reliable delivery via WS-Reliable Messaging [8]
- Fault tolerant data transport
- Support for different underlying transport implementations such as TCP, UDP, Multicast, SSL, RTP, HTTP.
- Discovery Service to find nearest brokers / resources.

4.2 NaradaBrokering for Reliable Delivery and Performance

Transferring data in a stream between components using direct connection has some disadvantages as given below

- Failure of the TCP connection or any of the nodes in the TCP link causes data loss and the system may have to restart the data flow for the computation to begin.
- Intermittent disconnects of a component or component failure would terminate the data transfer which may have undesired effects on the computation. Typically, we would expect the computation to resume from the point where it was obstructed.

NaradaBrokering helps us to alleviate the above problems. NaradaBrokering features reliable delivery at the routing substrate level. Reliable message delivery is also guaranteed during prolonged client disconnects and link, broker or reliable storage failure. NaradaBrokering will enable us to support message level security and fault tolerance compatible with Web Service standards. Message queuing may be used to ensure time synchronization and efficient cascading of data streams through multiple filter levels.

Ref [9] provides a detailed analysis of latency and performance metrics of the NaradaBrokering system.

4.3 HPSearch

We have been developing HPSearch [10] as an extension to an existing scripting language that binds Uniform Resource Identifiers (URI) [11] to the scripting language. In future work we may also provide support for WSAddressing [12] to refer to resources. This would allow us to access URI's either as variables or through a search interface (For example, to the Google web service). Every data source, filter or sink is identified by a URI on the web. In our case we use a scripting environment to create objects that can be used to access a variety of data sources. This is known as *"Binding URI to the Scripting Language"*. We have the following URI bindings in HPSearch.

- Reading from (or writing to) files (using `file://`)

- Reading files via http or ftp protocols (`http://` or `ftp://`)
- Reading from (or writing to) a raw socket (`socket://ip:port`)
- Reading from (or writing to) a topic in a brokering system (`topic://`)
- Reading from a database and streaming the results of SQL queries as XML (`jdbc:`). We have implemented a simple scheme to map the result set to XML[1].

The HPSearch shell uses NaradaBrokering (Section 4.1) to route data streams. In effect, processing is done by passing data through various programs that are accessible as services, in a Pipe-Filter fashion. HPSearch is designed as a scripting interface [13] to the Internet (Grid). We currently use the Rhino [14] implementation of Javascript, although any other scripting language like Python or Perl can also be used. Scripting poses numerous advantages as observed by Ref [15]. Rhino further allows us to define custom Javascript objects called as *host-objects* that help to dynamically access the host system. This feature can be useful to create objects that help manipulate data streams. Ref [16] illustrates the use of a proxy based Web Service that encapsulates the management and processing (filtering) of data streams. The overall application, which is made up of numerous such data filters is then controlled via HPSearch objects.

4.4 WSProxy to implement Data Filters

We make use of the WSProxy [17] to implement data filters. WSProxy is a Web Service based Proxy that can be invoked using simple Web Service calls. The WSProxy uses NaradaBrokering to route the data streams. WSProxy wraps the data streams obtained by subscribing and publishing to NaradaBrokering topics and makes them available to the filter application as simple input/output streams. The filter application reads data from the input stream and writes the filtered data to the output. This data is then sent to the next entity (filter, if available or HPC cluster) in the chain.

The WSProxy currently includes functions to set the parameters of the filter application. The filter may be augmented by functions to modify the filter parameters or set new parameters at run-time. This parameter setting functionality may be used by the HPC cluster to change the filtering semantics of the filter application.

The data transfer starts, once the initial protocol negotiation is done. The filters receive data in a stream, process it and write out the data to the next entity in the chain. We use HPSearch (Refer Section 4.3) scripting interface to initialize the disparate components and connect them in a data flow. The Crisis Grid flow prototype[2] (Refer Section 2.3) illustrates how we can connect components together in a dataflow.

Further, WSProxy may be programmed to catch errors and send notifications to the dataflow controlling engine (HPSearch `RequestHandler`). A user may then define custom actions using HPSearch - Javascript which include handling application specific errors, processing notifications and discover / rediscover resources among other actions.

In the event that a particular data source fails to deliver the data. the computation must be suspended and resumed later when the data delivery resumes. This feature can be coded using HPSearch-Javascript by having the filter send a notification to the data flow engine that the data stream is un-available. The workflow engine in turn may chose to suspend all the other components and then resume all the components once the failed data source is up and running.

5 High Performance Data Transport Protocol

Protocols such as SABUL [18], UDT [19] can dramatically improve the data transport performance. SOAP-based Web Services have a huge impact on performance, related to increase in size of XML data and the additional *CPU* resources required to parse XML data. To improve performance of SOAP-based Web Services, CSV (Comma Separated Values), HDF (Hierarchical Data Format) or binary formats such as DIME (Direct Internet Message Encapsulation) [20] may be used. Another approach is to replace XML with ASCII delimited fields or compress XML [21].

[1]Refer http://www.hpsearch.org/notes/scripting
[2]`http://www.hpsearch.org/demo/crisisGridFlow`

5.1 Transport Binding

Transport binding [22] allows us to exploit any applicable high performance transport mechanism at any level in the OSI stack. We negotiate the control mechanism using a simple model such as SOAP-HTTP. This step establishes the details of the high performance transport. After this initial negotiation, the data is streamed in an optimal fashion, possibly using a different encoding and a different transport protocol. This streaming uses a different port for the initial negotiation so we can still change the stream processing using out of band SOAP messages. This approach is similar to the one used by WS-Secure Conversation [23] which is used to establish and share security context to enable a secure conversation.

Specifically we propose a scheme to implement MPI-IO [24] using NaradaBrokering as a reliable routing substrate. We prefer a hybrid model where MPI - *optimized for low latency and high bandwidth* - is used internally for data transfer and NaradaBrokering is used externally. However, there will be a need to interoperate with distributed versions of MPI such as MPICH-G2. We initialize a stream of MPI messages containing SOAP and NaradaBrokering header information. A unique integer (UUID) is generated at this phase and used to label the stream and the mapping of NaradaBrokering/SOAP to MPI. Further we only need this integer, stream continuation tag, and a sequence number on each MPI message in the high performance stream. Thus after initial negotiation the MPI messages can be transferred with no performance degradation.

Security, reliable messaging, routing and performance aspects of all data transport are controlled at the application layer using SOAP compliant messaging infrastructure implementing a high performance controllable overlay network. These features are implemented at the various levels of the OSI stack and can use the best available protocols.

The overlay network transports streams, which are ordered sets of messages. The message abstraction includes a header and a body where the header is transported in a lightweight fashion allowing the overlay network to easily process the header information. Further, we map all the relevant header (control) information to the header of the overlay network so that the control information can be interpreted with low overhead at any intermediary or endpoint node that needs to act on the message.

The overlay network is implemented using NaradaBrokering messaging. It is linked to the Web Services using a SOAP handler or equivalent mechanism. The handler handles message control using control instructions in the SOAP header. The SOAP header is carried in the header (termed *synopsis*) of each NaradaBrokering message in a fashion that depends on the particular protocol. For example

- In a standard HTTP message, the synopsis holds the full XML of the SOAP header
- In an RTP event, the header is a short binary string carrying a UUID mapping to the binary protocol type, sequence number and stream position (begin, end, continue) and SOAP header used in negotiating the stream transport.

Thus while processing streams of messages, the full details of SOAP header are used only in the initial negotiations but not in individual messages that comprise the stream. These negotiations relate to achieving agreement on details of transport, including protocol encoding and routing. Individual messages thereafter contain information regarding the session, stream, source and numbering information corresponding to the message within the sequence. Additional information exchanged during the aforementioned SOAP negotiations can be retrieved at any time using this information contained in the individual messages.

5.2 High Performance Web Services

We plan to use Web Services and existing technologies for implementing data processing filters. In Section 5, we noted a few technologies that help us to achieve a dramatic improvement in data transfer throughput while Section 5.1 shows how we can integrate different transport level protocols in NaradaBrokering. We propose an entirely new approach for creating Web Services by separating the control channel from the data channel. With this approach, SOAP/XML can be used for the control channel while the data channel can employ a different protocol and packaging format such as WS-DMX (Web Services for Data Mining and eXploration) [25, 26, 27] which is based on the UDT protocol. Table 1 shows the relative performance using our implementation of WS-DMX called Open DMIX (Data Mining, Integration and eXploration). We compare the performance using

- SOAP/XML using TCP
- WS-DMX using ASCII records and UDT protocol for data channel

- WS-DMX using binary records and UDT protocol for data channel

Record	SOAP/XML			WS-DMX / ASCII			WS-DMX / Binary		
count	`MB`	μ	σ/μ	`MB`	μ	σ/μ	`MB`	μ	σ/μ
10000	0.93	2.04	6.45%	0.5	1.47	0.61%	0.28	1.45	0.38%
50000	4.65	8.21	1.57%	2.4	1.79	0.50%	1.4	1.63	0.27%
150000	13.9	26.4	0.30%	7.2	2.09	0.62%	4.2	1.94	0.85%
375000	34.9	75.4	0.25%	18	3.08	0.29%	10.5	2.11	1.11%
1000000	93	278	0.11%	48	3.88	1.73%	28	3.32	0.25%
5000000	465	7020	2.23%	242	8.45	6.92%	140	5.60	8.12%

Table 1: Comparison of use of SOAP/XML and WS-DMX for processing teraflows

Here `MB` represents the amount of data transferred in each test, μ is the average time for data retrieval in seconds while σ is the standard deviation of the time. Each test was performed 5 times and the tests were performed over a *1 Gb/s* link between Chicago and Amsterdam. Note that using WS-DMX is over 1200 times faster than SOAP/XML when moving 5,000,000 data records.

6 Future Work

We have a prototype of the WSProxy that manages data streams using NaradaBrokering as the routing substrate. Currently, we can use the HPSearch-Javascript based workflow engine to setup simple data flows and connect the various filters together. The shell must be enhanced by adding capabilities to negotiate data transport setup thus effectively creating a scripting interface to orchestrating NaradaBrokering capabilities.

6.1 Implementing high performance data transfer using NaradaBrokering

NaradaBrokering already supports many important protocols such as UDP, TCP and parallel TCP and includes network monitoring and firewall penetration among other features. Ref [28] illustrates how GridFTP can be enhanced using NaradaBrokering as a reliable routing substrate. We plan to add similar support for MPI-IO as illustrated in Section 5.1. This strategy will be used in generating conventional MPI messages at the stream parallel computing interface and in supporting Grid enhanced MPI systems.

6.2 Changes to HPSearch prototype

Services allow querying them to get specific service data or resource properties (WS-Resource Properties [29]). This would allow the workflow engine to dynamically query the properties of a filter to help control the filtering semantics. The WSProxy based service currently implements a simple query scheme on obtaining the current status of the service (running, suspended, stopped). Other service specific data querying would be included in the future.

HPSearch allows us to define custom actions using scripts. We also plan to implement a more sophisticated notification system to handle errors as well as notifications [30] and operations for rediscovery and subsequent handling of resources.

Security [31] is paramount to the functioning of any system. Although the shell and WSProxy do not currently implement any security schemes, we plan to use the security features provided by NaradaBrokering [32]. We consider security in two parts.

- By implementing NaradaBrokering security features we can secure topics and the data sent on them. The WSProxy may be sent special security tokens / certificates which should be used when the WSProxy subscribes or publishes data on a particular topic.

- The invocation of WSProxy takes place as a normal Web Service. We use SOAP messaging for this purpose. This communication can be secured by using policies defined using WS-Security [33].

7 Conclusion

In this paper we discuss how we can combine the reliability features of the NaradaBrokering substrate along with the scripting interface of HPSearch to utilize streaming data in high performance computing applications. The paper addresses the issue of generating fault tolerant data streams that can be combined in a *Service-Oriented* architecture to process data from real-time data sources. By binding existing high performance transport protocols to NaradaBrokering in a transparent manner, we can achieve more throughput at the data transport level. We believe that, by off-loading the data filtering work to service based filters, we can minimize unnecessary network traffic and unwanted or insensitive data in computation thereby realizing more throughput and hence a more cost effective solution.

References

[1] Harshawardhan Gadgil, Jin-Yong Choi, Bernie Engel, Geoffrey Fox, Sunghoon Ko, Shrideep Pallickara, and Marlon Pierce. *Management of data streams for a real time flood simulation.* Submitted to 5th IEEE/ACM International Workshop on Grid Computing.

[2] *Global MMCS.* Project page http://www.globalmmcs.org.

[3] Ahmet Uyar, Wenjun Wu, Hasan Bulut, and Geoffrey Fox. *An Integrated Videoconferencing System for Heterogeneous Multimedia Collaboration.* 13-15 Honolulu, Hawaii, USA, August 2003. 7th IASTED International Conference on Internet and Multimedia Systems and Applications, IMSA.

[4] K. Amin, M. Hategan, G. von Laszewski, A. Rossi, S. Hampton, and N. J. Zaluzec. *GridAnt: A Client-Controllable Grid Workflow System.* 37th Hawaii International Conference on System Science, Island of Hawaii, Big Island, 2004. 5-8 Jan, Available: http://www.mcs.anl.gov/˜gregor/papers/vonLaszewski–gridant-hics.pdf.

[5] Gridftp protocol. Project page: http://www.globus.org/datagrid/gridftp.html.

[6] Geoffrey Fox Shrideep Pallickara. *NaradaBrokering: A Distributed Middleware Framework and Architecture for enabling Durable Peer-To-Peer Grids.* Rio Janerio, Brazil, June 2003. ACM/IFIP/USENIX International Middleware Conference.

[7] *NaradaBrokering.* Project Page: http://www.naradabrokering.org.

[8] *Web Services Reliable Messaging.* http://www-106.ibm.com/developerworks/webservices/library/ws-rm/.

[9] Geoffrey Fox, Shrideep Pallickara, and Xi Rao. *A Scaleable Event Infrastructure for Peer to Peer Grids.* Seattle, WA, 2002. ACM Java Grande ISCOPE Conference. pp 66-75.

[10] *HPSearch.* Refer: http://www.hpsearch.org.

[11] T. Berners-Lee et.al. *Uniform Resource Identifiers.* IETF RFC, http://ftp.ics.uci.edu/pub/ietf/uri/rfc2396.txt.

[12] *WS-Addressing.* Specification available http://www-106.ibm.com/developerworks/library/specification/ws-add/.

[13] Geoffrey Fox, Harshawardhan Gadgil, and Shrideep Pallickara. *HPSearch and NaradaBrokering: Workflow Scripting and Stream Management.* Edinburgh, Dec 2003. Presentation available at http://grids.ucs.indiana.edu/ptliupages/publications/ presentations/edinburghworkflowdec03.ppt.

[14] Rhino: Javascript for java. Project page: http://www.mozilla.org/rhino.

[15] John K. Ousterhout. *Scripting: Higher Level Programming for the 21st Century.* http://home.pacbell.net/ouster/scripting.html.

[16] *Crisis Grid flow Prototype.* Prototype available at http://www.hpsearch.org/demo.

[17] *WSProxy.* Refer: http://www.hpsearch.org/notes/wsproxy/.

[18] R. L. Grossman, M. Mazzucco, H. Sivakumar, Y. Pan, and Q. Zhang. *SABUL - Simple Available Bandwidth Utilization Library for high-speed wide area networks.* To appear in Journal of Supercomputing, 2003.

[19] Yunhong Gu and Robert Grossman. *Using UDP for Reliable Data Transfer over High Bandwidth-Delay Product Networks.* Available at http://www.ncdm.uic.edu/papers/udt-protocol.pdf.

[20] Henrik Frystyk Nielsen, Henry Sanders, Russell Butek, and Simon Nash. *Internet-draft: Direct Internet Message Encapsulation.* Retrieved from http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-0%2.txt, 2002.

[21] Robert A. van Engelen. *Pushing the soap envelope with web services for scientific computing.* International Conference on Web Services (ICWS), 2003. pages 346–354.

[22] Geoffrey Fox. *High Performance and Specialized Transport in Web Services*, July 2004. Available from http://grids.ucs.indiana.edu/ptliupages/publications/WS-TransportBinding.doc.

[23] *Web Services Secure Conversation Language.* Available at: http://www-106.ibm.com/developerworks/library/specification/ws-secon.

[24] *Message Passing Interface - IO.* Refer: http://www.mpi-forum.org.

[25] Robert L. Grossman et. al. *Teraflows over Gigabit WANs with UDT.* To appear in Journal of Future Computer Systems, Elsevier Press.

[26] Yunhong Gu, Xinwei Hong, and Robert Grossman. *Experiences in Design and Implementation of a High Performance Transport Protocol.* To appear in SC 04.

[27] Robert L. Grossman et. al. *Experimental Studies of High Performance Web Services For Distributed Data Mining.*

[28] Sang Boem Lim, Geoffrey Fox, Shrideep Pallickara, and Marlon Pierce. *Web Service Robust GridFTP.* Las Vegas, June 2004. The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA04).

[29] *WS-Resource Framework.* Refer: http://www.globus.org/wsrf/.

[30] *Web Services Notification.* Available from http://www-106.ibm.com/developerworks/library/specification/ws-notification/.

[31] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. *A Security Architecture for Computational Grids.* 5th ACM Conference on Computer and Communications Security Conference, 1998. pp. 83-92.

[32] Yan Yan et. al. *Implementing a Prototype of the Security Framework for Distributed Brokering Systems.* International Conference on Security and Management, 2003.

[33] B. Atkinson et. al. *Web Services Security (WS-Security) Version 1.0.* 5 April, 2002, Available from http://www-106.ibm.com/developerworks/library/ws-secure.