A Portal Based Approach to Viewing Aggregated Network Performance Data in Distributed Brokering Systems

Gurhan Gunduz[1,2], Shrideep Pallickara[2] and Geoffrey Fox[2]
(ggunduz@syr.edu), Department of Electrical Engineering and Computer Science, Syracuse University.[1]
(spallick,gcf@indiana.edu) Community Grid Labs, Indiana University.[2]

1.0 Introduction

Distributed messaging/brokering systems which provide a scalable infrastructure for several applications involve a set of nodes (called brokers) communicating with each other. Clients connected to any of these brokers can communicate with each other.

Effective utilization of networks and responsiveness to changing network conditions are essential to mitigate network utilization problems associated with distributed messaging systems. This is predicated on four essential issues. First, there needs to be a monitoring of the communication links hosted at a broker node. This will involve monitoring the links to other brokers and of course to clients connected to the broker in question.

Second, brokers should be able to able to respond to changing local and remote conditions. To respond to changing local conditions, a broker may deploy different transport protocols. For example, if there is a high concentration of clients from a given geographic location, a broker may choose to deploy multicast (if possible) for communications with these nodes. To enable a broker to respond to changing remote conditions there should be a scheme for aggregating network performance from individual broker nodes. These aggregator nodes thus snapshot the state of the network domain within which it aggregates performance.

Third, it should be possible to view the aggregated network performance data and specify constraints to detect network thresholds under which remedial measures might need to be initiated. These measures could involve creation/purging of connections, migration of underlying transports for communications and forcing connected clients to connect to another broker.

Finally, in a truly dynamic system, the detections and responses would be initiated dynamically. Such self healing systems can respond, in real-time, to changing local and remote network conditions. These systems are responsive to changing client concentrations and also optimize bandwidth utilizations associated with the interactions that they route. However, it should be noted that a truly dynamic system still would need performance monitoring at a broker node, require this broker to respond to changing local conditions and remote conditions based on performance accumulated at an aggregator node.

In this paper, we present a scheme to view aggregated performance information inside a portal and specify constraints on the accumulated performance information to detect network conditions, based on specified thresholds on measured performance factors. This information would then be used to work around network bottlenecks and also be used to drive heuristics for generating optimal routes (and transport protocols deployed) to reach a set of computed destinations. The scheme that we have prototyped in our current work provides for detection of network conditions. The remedial measures that are initiated are currently based on static techniques. We suggest that this work is a precursor to a dynamically responsive system. This work builds upon our previous work [1] involving the design of a framework for aggregation network performance. The investigations that we report in this paper are in the context of NaradaBrokering [2-10] a distributed brokering system.

The remainder of this paper is organized as follows. In section 2.0 we discuss related work in the areas relevant to this paper. In section 3.0 we present a brief overview of the NaradaBrokering system and our framework for aggregating network performance. In section 4.0 we discuss why we adopted a portal based approach to viewing aggregated network performance data and outline details pertaining to the incorporation of performance data within the portal. Section 5.0 presents results from our experiments. Finally we discuss the future work and conclusions derived from the work outlined in this paper.

2.0 Related Work

The Network Weather System (NWS) [11,12] is one of the most well-known systems in network monitoring. NWS periodically monitors resources and dynamically forecasts the short-term performance. Sensors in NWS collect end-

to-end TCP/IP performance, available CPU percentage and available non-paged memory information and send them to a specific network location at regular intervals. In Refs [13, 14] congestion and bandwidth of the links are measured non-intrusively by actively probing the network between designated hosts using bprobe and cprobe tools repectively. IP Provider Metrics, a subgroup of IETF's Bench Marking Working Group (BMWG), is trying to develop a set of standard metrics that can be applied to the quality, performance and reliability of Internet data delivery services [15]. Cooperative Association for Internet Data Analysis (CADIA) [16] provides and analyses measurement tools currently available.

XQuery [17], XQL [18], XML-GL [19], XML-QL [20] are query languages for XML. The XML Query Language (XQL) is designed specifically for XML documents and provides a single syntax that can be used for queries, addressing and patterns. XQuery is a language that is designed for processing XML data and also data whose structure is similar to XML. XQuery is derived from XML query language Quilt [21], which in turn borrowed features from XPATH [22], XQL, XML-QL and SQL [23]. XML-GL is a graphical query language for XML data and it can query multiple documents like XQuery and XML-QL.

In the portal domain, there are number of commercial (IBM Websphere portal [24], BEA Weblogic portal [25], Plumtree[26]) and non commercial(Apache Jetspeed Enterprise Information Portal [27]) web portal solutions available.

3.0 NaradaBrokering and the Performance Aggregation Framework Overview

NaradaBrokering is a distributed event brokering system designed to run on a large network of cooperating broker nodes. Broker nodes can run on separate clients, whether these clients are associated with users or resources. Broker nodes are organized in a cluster-based architecture which allows the system to support large heterogeneous client configurations that scale to arbitrary size. NaradaBrokering provides support for centralized, distributed and peer-to-peer (P2P) [28] models. Communication within NaradaBrokering is asynchronous and the system has been deployed to support audio/video conferencing systems.

The NaradaBrokering transport framework [29] facilitates easy addition of transport protocols for communications between NaradaBrokering nodes. One of the most important elements in the transport framework is the Link primitive, which encapsulates operations between two communications endpoints and abstracts details pertaining to communications and handshakes. Currently TCP, UDP, RTP, Multicast, SSL and HTTP based implementations of the transport framework exist.

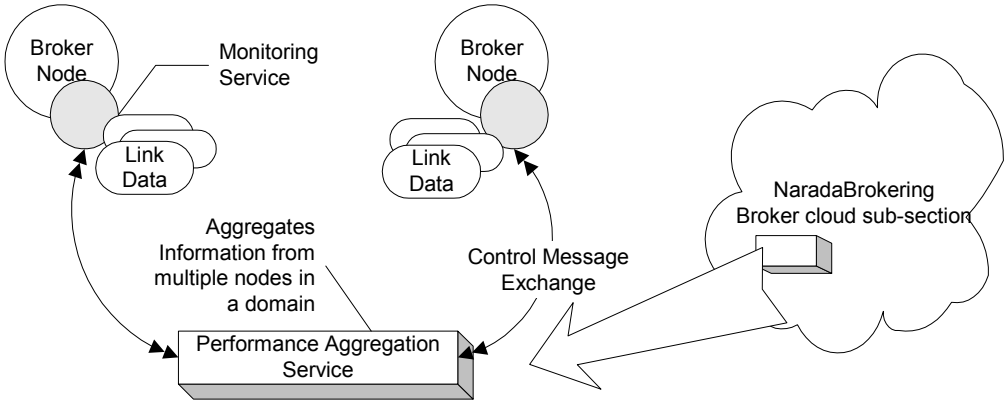

Figure 1: Aggregation Service Overview

In the performance aggregation framework [1], every broker in NaradaBrokering incorporates a monitoring service (as shown in Figure 1) that monitors the state of the links originating from the broker node. An implementation of the Link primitive can measure and report performance factors such as *bandwidth*, *jitter*, *transit delays*, *loss rates* and *system throughputs*. The Monitoring Service cycles through this list of links at regular intervals to retrieve performance information from each link. The Monitoring Service then reports these retrieved information to the Aggregation Node. Each Aggregation Node aggregates performance data from monitoring services running at multiple nodes. Each link has a unique ID associated with it which allows us to identify the links being the monitored in the aggregated system.

The monitoring service that runs at every node encapsulates performance data gathered from each link in an XML structure. This performance data is then sent to the Aggregation Node, which stores this data. There are two different approaches that could be used to store aggregated performance metrics. The first approach would be to use of flat files to store performance metrics. We are using this approach in our current implementation. Even though it is easy to implement this approach, search could be a problem in large files since all the data is needed to be searched sequentially. Another approach is to use a lightweight XML database. This approach requires extra software but it has advantages of database systems, like efficient searches. We are investigating the issues such as efficiency, time required for queries and storage space in the context of lightweight XML databases.

4.0 Aggregated Performance and Information Portals

Once data is accumulated at an aggregated node, this data needs to be available to interested entities. There are two important issues pertaining to the access to this data. First, there is the issue of easy access to this data. Users should be able to access this aggregated information from disparate locations. Enabling access to this information through browsers (which are now available even for hand-held devices) would be an excellent choice while also obviating the need to develop proprietary solutions to view the data.

Second, a need arises to present only the relevant parts of the aggregated information to interested/authorized users. For various reasons some of the aggregated information needs to be viewed only by a user with administrative privileges. Also, some users may need information pertaining to the broker nodes that they manage. This calls for presenting different views of the aggregated data to different users.

Portals fill this need of easy accessibility and restricted views to the same data sets. Portals can collect content from disparate sources such as HTML, XML and images into one page. This provides an integrated view of different data sets while being easy to use. Portals can be accessed from anywhere at any time using web browsers. Portals also provide personalized and customized user/group views. This feature allows us to restrict access to network performance data by ensuring different users or groups have access to different features of the aggregated performance data. For example, while administrators may have access to entire performance data, some other users may have an access to only usage patterns or specific metrics or a subset of the aggregated performance data.

Even though early portals started as search engines that provided access to a greater amount of information, they have added more content, services and personalization while continuing to grow rapidly. There are four portal categories – Corporate or Enterprise (intranet) portals, e-business extranet portals, personal (WAP) portals and public or mega (internet) portals. A Portal's disadvantage stems from the need of a dedicated administrator to maintain it. Furthermore, access to information using portals might be slower than proprietary solutions to access the data set.

4.1 Accumulation of data inside a portal

Users can access the information, accumulated within the aggregators, via a *portlet* residing in a Portal. A portlet is a specialized module, a Java servlet [30], which operates inside a portal. A portlet and a servlet are the same thing, where servlet is an application within a Web server and a portlet is an application within a portal. Portlets can contain variety of information such as sports scores, world news and stock quotes. Portlets interact with web clients indirectly through portals using a request/response paradigm implemented by the portlet container (HTTP). Multiple portlets can be assigned to one or more portals. Portlets are stored in portlet catalog where users can browse and add them into a desired location in their own portal pages and configure them to display personalized content. Portlets are developed, managed and displayed independent of other portlets. Portals provide API's for portlet creation.

There are many commercial portal environments available from companies such as IBM, Sun, Microsoft and BEA Systems Inc. Beside those there are also free and open source portal implementations available. We are currently using Apache Jetspeed Enterprise Information Portal as our portal environment. We chose Jetspeed because it is free and open source, which makes custom modifications possible.

4.2 Viewing the aggregated performance data

Since the information that we store is in XML format, we need to transform it into an HTML to display it in the portal. The Extensible Stylesheet Language (XSL) [31] is a transformation and formatting language for XML documents. XSL stylesheet is used to express designers' intention about how the content should be styled, laid out

and paginated onto some presentation medium such as a window in a web browser or a hand held device or a set of physical pages in a catalog, report or book. XSL consists of three parts – XSLT [32] (language for transforming XML document), XPath (for querying purposes) and XSL formatting objects (a vocabulary for formatting XML documents).

There are two approaches that can be deployed to view our aggregated performance data in the portal environment. First, we could use the XSLT portlet, which comes with the Apache Jetspeed. The XSLT portlet provides a service to convert a given XML file into HTML using the given XSL style sheet. The desired XML and XSL files can be specified by changing the portlet properties file. The second approach is that of writing a customized portlet using the Portlet API provided by Apache Jetspeed. A customized portlet can be used for more specific tasks. For example, it can be customized to gather information from a lightweight XML database, which we might use in our future implementations. In our investigations we have relied on using the XSLT portlet for processing information.

In one of our earliest experiments we measured the time required to initialize the XSLT portlet and to transform an XML file into an HTML file using the XSL file provided. We saw that the initialization time is under 250 microseconds and it is not related to the size of the file that it is transforming. Transformation time however increases proportionally with the size of the XML file (number of entities).

Furthermore, it should be noted that if the same user or a user from the same group tries to use that portlet, there is no initialization time and transformation time required. This is because of one of the features of Jetspeed where it caches the viewed information. Since the Jetspeed cache is refreshed at different time intervals, strategies for optimizing accesses need to take these cache invalidations into account and pre-fetch data to reduce loading times.

4.3 Detecting conditions

We choose to evaluate constraints in the aggregation node to reduce computation overheads at the broker nodes, which besides supporting data exchanges over links are also responsible for coordinating the measurement of performance factors. The broker is also responsible for dealing with other functions such as processing different types of interactions, computing destinations associated with interactions and finally efficient paths to reach these destinations.

As discussed in section 2.0, there are several languages available to query XML document. We are currently using XPath to query our XML documents. XPath is a W3C standard and is a language for addressing parts of an XML document. It uses path expressions to identify nodes in an XML document and also supports XML namespaces. XPath also provides basic facilities for manipulation of Strings, numbers and Boolean values.

There are three reasons why we choose XPath for our detections. First, XPath is a W3C standard and as a result there are mature implementations available for use. Second, XPath is also a part of XSL which is used to present the documents. Finally, we are not currently limited, in any way, in our ability to specify sophisticated constraints efficiently.

We query the XML file using XPath to detect network conditions from the aggregated performance data. All the metrics, *bandwidth*, *jitter*, *transit delays*, *loss rates* and *system throughput* can be checked for thresholds. If a certain threshold is reached then the broker node can be informed to take actions to correct the situation. For example broker node would reject new connections or cancel some of the links.

We have measured the time required for querying the information related to a link. The average time to evaluate a query is between 2-4 milliseconds depending on the complexity of the query. This evaluation had it taken place at the link itself, would have been unacceptable (for most of the links), since links have communication delays that are in the range of 500-700 microseconds per communication hop. We are currently working on optimized query methods to reduce the evaluation time.

5.0 Monitoring and Portal Overheads

In this section we report on our experiments. The experiments were performed on a Windows 2000 machine (Pentium-4, 1.5 GHz, 512 MB) and all the processes involved ran using the Java 1.4 JRE VM. In our experiments we simulate the presence of multiple links (sometimes reaching several thousands) since it would be infeasible to

actually set up the large number of links that we deal with in our experiments. Also, in our experiments the portal is running on the same machine where the database file is on. There is thus no network delay to load the database file.

For our experiments we did 3 different kinds of measurements, First, we measured the time for constructing the W3C Document Object Model from an XML file. The time associated with this construction varies with the number of XML elements in the file, which corresponds to the number of links whose information has been aggregated. Figure 2 depicts the W3C Document creation time as a function of the number of elements in the XML file. The cost varies from 691 milliseconds for an XML file with 100 elements (representing 100 links) to 2.824 seconds for an XML file with 16000 entries.

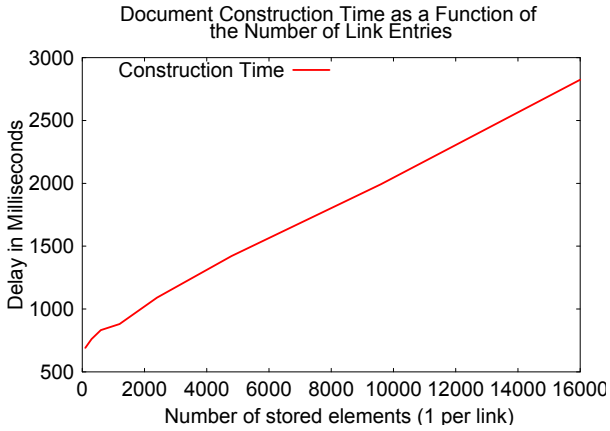

Figure 2: Document Construction time

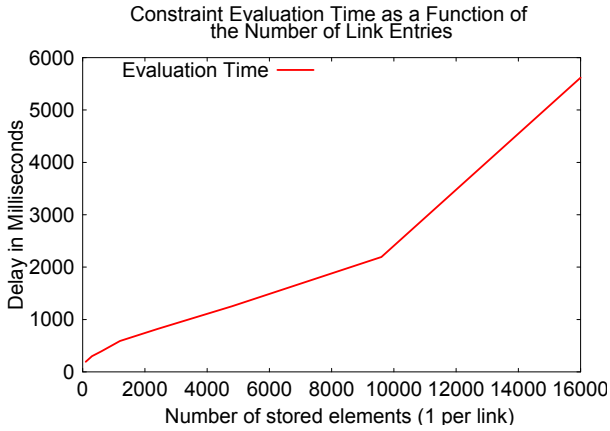

Figure 3: Query Evaluation time

Second, we measured the time to evaluate typical queries based on different number of links in the aggregated data. The graph is Figure 3 depicts this increase in evaluation time corresponding to increases in the number of entries in the XML flat-file. The evaluation operation can of course be optimized considerably by evaluating queries only for those links (each with a unique link ID) whose values have changed. In our earlier work we had incorporated a scheme where monitoring nodes would report data about links, only if there is a certain percentage change from its last reported data. This optimization would allow us to reduce query evaluation time very significantly.

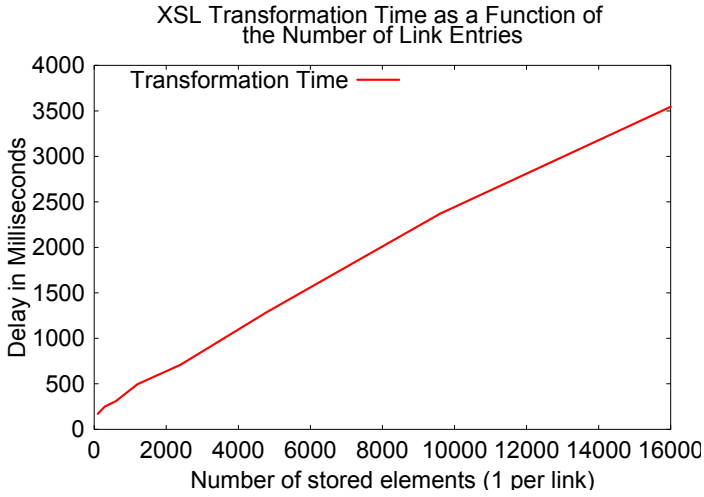

Figure 4: XSL Transformation time

Finally, we include timing delays associated with time it takes the XSLT portlet to transform an XML file into the corresponding HTML file using a given XSL style sheet. Figure 4 depicts the time needed for this transformation.

6.0 Conclusions and Future Work

In this paper we outlined a strategy to incorporate XML-encapsulated network performance data into portals. We outlined pros and cons of doing so, and the strategy we deployed to achieve this. Finally, we also included performance numbers from our prototype implementation.

Trade-offs of using flat files versus light-weight databases, including issues pertaining to efficiency, storage space, and search time will be evaluated. Optimization in the mining of data is another area that needs to be researched further. Finally we intend to incorporate this information pertaining to the identification of system bottlenecks to aid routing algorithms to assuage any network degradations that may exist within the messaging infrastructure.

Individual brokers maintain weighted broker network maps (BNMs) that are used to compute routes to reach destinations. Aggregated information can be used to dynamically update the costs associated with traversal. Similarly, certain routes can be invalidated as a result of certain network conditions. We are now well poised to investigate the effect on routing decisions and accompanying optimal routes to reach computed destinations as a result of detection of network degradations in certain sections of the network fabric.

References

1. [nb-sec] A Framework for Aggregating Network Performance in Distributed Brokering Systems. Gurhan Gunduz, Shrideep Pallickara and Geoffrey Fox. (To appear) Proceedings of the 9th International Conference on Computer, Communication and Control Technologies
2. The NaradaBrokering System http://www.naradabrokering.org
3. A Middleware Framework and Architecture for Peer-to-Peer Grids. (To appear) Proceedings of ACM/ IFIP/USENIX International Middleware Conference Middleware-2003. Shrideep Pallickara and Geoffrey Fox
4. NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids. Geoffrey Fox and Shrideep Pallickara. Chapter 22 of "Grid Computing: Making the Global Infrastructure a Reality". John Wiley April'03.
5. The Narada Event Brokering System: Overview and Extensions. Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, June 2002. pp 353-359.
6. A Scaleable Event Infrastructure for Peer to Peer Grids. Geoffrey Fox, Shrideep Pallickara and Xi Rao. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington. November 2002.
7. "JMS Compliance in the Narada Event Brokering System." Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Internet Computing (IC-02). June 2002. pp 391-402.
8. Grid Services for Earthquake Science. Fox et al. Concurrency & Computation: Practice & Experience.14(6-7):371-393.
9. "Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service". Bulut et al. Proceedings of the IASTED International Conference on Communications, Internet, and Information Technology, 2002, in US Virgin Islands.
10. An Event Service to Support Grid Computational Environments. Geoffrey Fox and Shrideep Pallickara. Journal of *Concurrency and Computation: Practice & Experience*. Volume 14(13-15) pp 1097-1129.
11. R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The Network Weather Service. Tech. Rep. TR-cs97-540, University of California, San Diego, May 1997.
12. R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. Proceedings of the 6th IEEE Symp. On High Performance Distributed Computing, August 1997.
13. R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report TR-96-007, Boston University 1996.
14. R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical Report TR-96-006, Boston University 1996.
15. IETF Benchmark Working subgroup: http://www.ietf.org/html.charters/ippm-charter.html
16. CAIDA http://www.caida.org/tools/
17. *XQuery 1.0: An XML Query Language*, W3C Working Draft (16 August 2002), see http://www.w3.org/TR/xquery.
18. Robie, J.; Lapp, J.; Schach, D. (1998). *XML Query Language* (XQL). In: Marchiori, M. (ed.): QL'98 --- The Query Languages Workshop. W3C. http://www.w3.org/TandS/QL/QL98/pp/xql.html
19. S. Ceri, S. Comai, E. Damiani, P. Fraternali, and S. Paraboschi. XML-GL: a graphical language for querying and restructuring XML documents. In Proc. of the Int. World Wide Web Conference, Canada, 1999.
20. Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. *A Query Language for XML*. See http://www.research.att.com/~mff/files/final.html
21. Don Chamberlin, Jonathan Robie, and Daniela Florescu. *Quilt: an XML Query Language for Heterogeneous Data Sources*. In *Lecture Notes in Computer Science*, Springer-Verlag, Dec. 2000. Also available at

http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf. See also http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html.
22. World Wide Web Consortium. *XML Path Language (XPath) Version 1.0*. W3C Recommendation, Nov. 16, 1999. See http://www.w3.org/TR/xpath.html
23. International Organization for Standardization (ISO). *Information Technology-Database Language SQL*. Standard No. ISO/IEC 9075:1999. (Available from American National Standards Institute, New York, NY 10036, (212) 642-4900.)
24. IBM Websphere Portal http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/portal
25. BEA Weblogic Portal http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/portal
26. Plumtree Portal http://www.plumtree.com/products/midmarket/
27. Apache Jetspeed. http://jakarta.apache.org/jetspeed/site/index.html
28. [p2p-book]Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Edited by A. Oram. O'Rielly Press, CA. March 2001.
29. [nb-trans-fw] A Transport Framework for Distributed Brokering Systems. Shrideep Pallickara, et al.
30. [servlet] Java Servlet Technology http://java.sun.com/products/servlet/
31. [xsl] World Wide Web Consortium. Extensible Stylesheet Language (XSL). W3C Working Draft. See http://www.w3.org/TR/xsl/
32. [xslt] World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation. See http://www.w3.org/TR/xslt