GridFTP and Parallel TCP Support in NaradaBrokering

Sang Boem Lim[1], Geoffrey Fox[2], Ali Kaplan[2], Shrideep Pallickara[2] and Marlon Pierce[2]

[1] Supercomputing Application Technology Department at
Korea Institute of Science and Technology Information (KISTI)
P.O. Box 122, Yuseong, Daejeon, Republic of Korea
slim@kisti.re.kr

[2] Community Grid Labs, Indiana University
501 N. Morton St. Suite 224
Bloomington, IN 47404, USA
{gcf, alikapla, spallick, marpierc}@indiana.edu

Abstract. Many of the key features of file transfer mechanisms like reliable file transferring and parallel transferring are developed as part of the service. It makes very hard to re-use the same code for the different systems. We are trying to overcome this disadvantage by decoupling useful features of file transfer mechanisms from the implementation of the service and protocol, and instead placed into the messaging substrate. We may thus treat file transfer operations as a specific usage case for a more general messaging environment. This will allow us to provide file transfer quality of service to other file transfer tools that does not have same features.

1 Introduction

Today's network environments require people to download many things on a daily basis. Especially new technologies developed recently, like Grid environments, require reliable, secure high performance file transfer as the most important services. GridFTP [1] [17] is the one of the most common data transfer services for the Grid and is a key feature of Data Grids [2]. This protocol provides secure, efficient data movement in Grid environments by extending the standard FTP protocol. In addition to the standard FTP features, the GridFTP protocol supports various features offered by the Grid storage systems currently in use.

Even though GridFTP has good features of file recovery technologies,many interesting features of GridFTP are tied to its protocol and implementation. Providing these features to other file transfer services (such as those based on Web Services, for instance) requires reimplementation and re-engineering. These shortcomings may be addressed by inserting a reliable, high performance *messaging substrate* between the client and service. This addresses specific problems in GridFTP client lifetimes, but more generally will allow us to extend GridFTP-like features to other services without extensive reimplementation. Also GridFTP has a restriction that the client needs to remain active at all the times until the transfer finishes. This in turn implies that

we cannot use the rich set of recovery features of GridFTP when the client state has been lost. In the event of client state loss, transfer has to restart from scratch.

In this report we present our work that has addressed the client-active-at-all-times constraint. The remainder of this report is organized as follows. In section 2 we present an overview of related work. In section 3 we present a overview of the NaradaBrokering system and the services within NaradaBrokering. In section 4 we provide details regarding our work. In section 5 we present some benchmark results and its analysis. Finally in section 6 we present our conclusions and future work.

2 Related Works

We are using many different file transfer mechanisms on daily basis. One of the most commonly used file transfer mechanism is File Transfer Protocol (FTP) [5]. This is the simplest way to exchange files between computers. FTP is an application protocol that uses the TCP/IP protocols. A more secure replacement for the common FTP, protocol is Secure Copy (SCP), which uses the Secure Shell (SSH) as the lower-level communication protocol. From the popularities of World Wide Web, we are also commonly using Hypertext Transfer Protocol (HTTP) as mechanism for transferring files. Even though some of file transfer mechanisms are quite reliable, these mechanisms do not provide guaranteed, reliable file transfer features like automatic recovery from failures.

Issues about reliable file transfer mechanism are more actively discussed and developed from the Grid community recently. More relevant service to our project is Reliable File Transfer (RFT) [3] [4] service developed by the Globus. RFT service provides reliable file transfer mechanisms like automatic failure recovery. In the next section we will discuss more about behaviors of RFT.

2.1 Comparison with Reliable File Transfer

The RFT is developed with automatic failure recovery while overcoming the limitation of its predecessor technology, GridFTP. The most important idea added to the RFT service is automatic failure recovery mechanism when any problems are occurred during file transfer like dropped connections and temporary network outage. The RFT is dealing with problem by performing a retry until the problem is resolved. The RFT also will inherit all the features that GridFTP has since it is built on top of existing GridFTP. The RFT will inherit most of the automatic recovery features like restart support and remote problems of the RFT service and it also will not lose performance of GridFTP.

The RFT service resolved a strict restriction of its predecessor GridFTP. The client of GridFTP needs to remain active at all the times until the transfer finishes. However, the RFT no longer requires this restriction. The RFT introduced a non-user-based service. This service will store the transfer state in a persistent manner and this state will be used to recover transfer from the last marker recorded for that transfer when failure occurs including the client state failure.

The RFT service itself has significant features to make reliable data transfer. However, the RFT service is not portable to any other systems. Once again our main goal of decoupling reliable features from the implementation is to make a portable system that can be deployed

into any file transfer mechanisms and make that mechanism reliable by using NaradaBrokering as a middleware.

3 NaradaBrokering

NaradaBrokering [7] [8] is messaging middleware designed to run on a large network of cooperating broker nodes (we avoid the use of the term *server*s to distinguish it clearly from the application servers that would be among the sources/sinks to messages processed within the system). Communication within NaradaBrokering is asynchronous and the system can support large client configurations publishing messages at a very high rate. The system places no restrictions on the number, rate and size of messages issued by clients.

In NaradaBrokering entities can also specify constraints on the Quality-of-Service (QoS) related to the delivery of messages. Among these services is the reliable delivery service, which facilitates delivery of events to interested entities in the presence of node and link failures. Furthermore, entities are able to retrieve any events that were issued during an entity's absence (either due to failures or an intentional disconnect). The scheme can also ensure guaranteed exactly-once ordered delivery.

Another service, relevant to this paper, is NaradaBrokering's Fragmentation/Coalescing service. This service splits large files into manageable fragments and proceeds to publish individual fragments. Upon receipt at a consuming entity these fragments are stored into a temporary area. Once it has been determined (by the coalescing service) that all the fragments for a certain file have been received, these fragments are coalesced into one large file and a notification is issued to the consuming entity regarding the successful receipt of the large file. The fragmentation/reliable delivery service combination can be used to facilitate transfer of large files reliably. Access to these capabilities is available to entities through the use of QoS constraints that can be specified. This facilitates exploiting these capabilities with systems such as GridFTP.

We emphasize here that NaradaBrokering software is a message routing system which provides QoS capabilities to any messages it sends. The NaradaBrokering system may be the messaging layer between many different applications, such as Audio/Video conferencing [11]. The QoS features provided by the NaradaBrokering system are independent of the implementation details of the endpoint applications that use it for messaging. Thus applications do not need to implement (for example) reliable messaging.

Furthermore, NaradaBrokering provides capabilities for communicating through a wide variety of firewalls and authenticating proxies while supporting different authenticating-challenge-response schemes such as Basic, Digest and NTLM (a proprietary Microsoft authenticating scheme).

4 Enhancing GridFTP

On the previous papers ([9] [12]) we already described enhancing mechanisms. In this paper we will briefly describe enhancing GridFTP with NaradaBrokering. And we will focus more on how reliable mechanism works in the NaradaBerokering.

GridFTP and other file transfer mechanisms may already incorporate a number of reliability features on there implementation of service and protocol. However, the most important weakness of these architectures is all the great features can not be used outside of its own architecture. This means whenever people want develop new file transfer mechanism and if they want existing features of other mechanisms, they have to re-develop same features within the service implementation. It is our goal to show that these reliability features can be decoupled from the implementation of the service and protocol, and instead placed into the messaging substrate. This will allow us to provide file transfer quality of service comparable to GridFTP in other file transfer tools (such as normal FTP, SCP, HTTP uploads, and similar mechanisms).

Figure 1 is present the basic architecture of integration between GridFTP and NaradaBrokering. For initial testing we developed the router approach even though proxy approach is the more preferred method. Main difference of those two approaches is usage of NaradaBrokering *Agent A*. The router approach will use NaradBrokering *Agent A* as simple router to transfer requests to the remote server. Key to the proxy approach is the remote GridFTP server is simulated by the NaradaBrokering *Agent A*. Since NaradaBrokering Agent A is a simple router on the router approach, it is easier than the proxy approach to implement. However, the router approach also has disadvantages like we have to change the user application, even though change is minor and also requires some minor extensions to FTP/GridFTP client codes to communicate with NaradaBrokering *Agent A*. The client and server communicate solely with the agents on the edge of the broker cloud. For the GridFTP client point of view, NaradaBrokering *Agent A* is a server and NaradaBrokering *Agent B* is a client for GridFTP server point of view. The proxy approach is the preferred method since the GridFTP client code and user application do not have to change. All existing GridFTP code and user application can be used in our architecture without any changes once this method is implemented. Disadvantage of this approach is it is harder to implement and time consuming process since we have to create GridFTP server from the scratch.

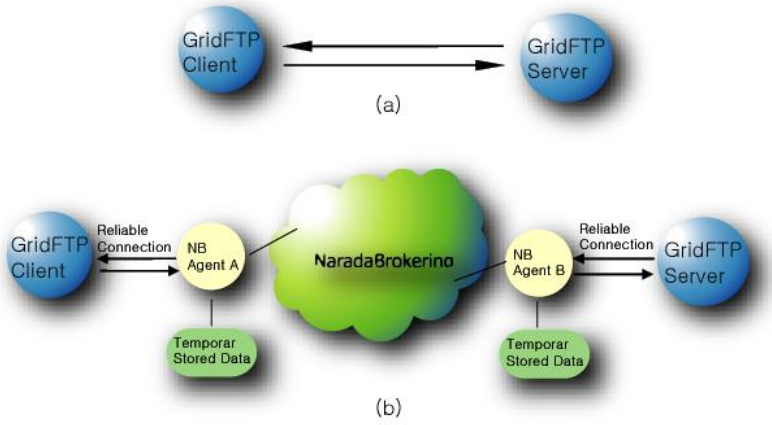

Figure 1 (a) Traditional GridFTP (b) GridFTP with NaradaBrokering

Currently, we have completed development of the uploading functionality of GridFTP with NaradBrokering using simple router approach. Connection between the GridFTP client and NaradaBrokering *Agent A;* and NaradaBrokering *Agent B* and GridFTP server are connected with a highspeed, reliable, possibly local, connection. This connection is needed because if connection between Grid FTP client and the NaradaBrokering *Agent A* is lost, we cannot recover from this failure. Recovering from this failure is out of scope (GridFTP is designed in

this way). All the data will be first transferred and stored into the temporary local space of NaradaBrokering *Agent A*. This temporary data will be used when any failure occurs inside of NaradaBrokering. Once all the data is stored locally in the NaradaBrokering *Agent A*, even if connection between GridFTP client and NaradaBrokering *Agent A* is lost, transferring to the server is guaranteed by NaradaBrokering. This feature is not on the current GridFTP system. In the current GridFTP system, if a client fails, the client has to begin uploading again from the start. NB Agent B also store data into the temporary local space. This temporary data will be used when any failure is occurred to the GridFTP server.

4.1 Reliable Mechanism in NaradaBrokering

We will describe in depth about how reliable mechanism of NaradaBrokering works. As we mentioned earlier we assumed that any of our architecture nodes could be go down during transfer except GridFTP server. To achieve this idea we are using acknowledgements and database. As we can see from Figure 2, the first step is that we divide large file into small pieces (a_1, a_2 … a_{n-1}, a_n) of same size except last piece that may truncated. Once NaradaBrokering get a piece from NaradaBrokering *Agent A*, it stores the piece into the database for any failure cases while NaradaBrokering is also sending same file to NaradaBrokering *Agent B*. An acknowledgment of receiving a piece on the NaradaBrokering from NaradaBrokering *Agent A* is taking place when NaradaBrokering is finished storing piece into the database. Also, there is an acknowledgment to NaradaBrokering after NaradaBrokering *Agent B* received and stored a piece into the temporary local directory. Those acknowledgments will be stored in the local file system and will be used when any failures occur during transferring a file. Once failure is fixed NaradaBrokering *Agent A*, and/or NaradaBrokering is looking for acknowledgment file and figure out the start point of resume transmission. For example, we have a machine failure on NaradaBrokering *Agent A* during sending a_7 with a_6 on acknowledgment file. After machine is re-started, NaradaBrokering Agent A is looking in the acknowledgment file and fined start point as a_7 since there are receive acknowledgment until a_6. This is goes to same between NaradaBrokering and NaradaBrokering *Agent B*.

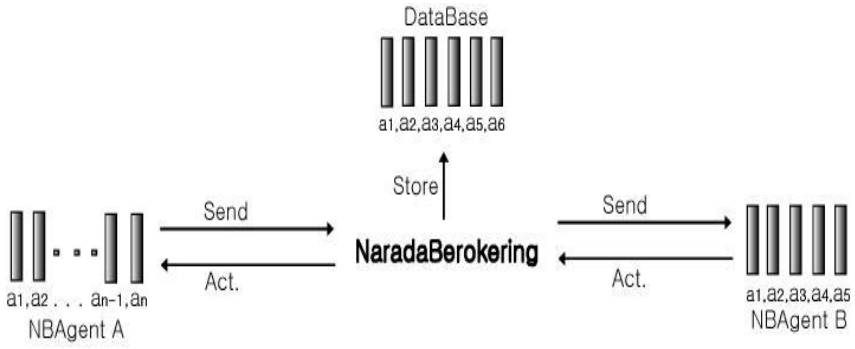

Figure 2 Reliable Mechanisms in NaradaBrokering

Database on the NaradaBrokering will be used as storage of small pieces of files. In this way we can transfer file from NaradaBrokering *Agent A* to NaradaBrokering without any guarantee of NaradaBrokering *Agent B* running and it is true for sending file form NaradaBrokering to NaradaBrokering *Agent B*. Even NaradaBrokering server itself can be go down. NaradaBro-

kering server is smart enough to know resuming point to NaradaBrokering *Agent B* after recovered from failure.

4.2 Multiple Stream Transfer Mechanism in NaradaBrokering

Advancement in network technologies is providing increasing data rates, but current TCP implementation prevents us to use maximum bandwidth across high-performance networks. This problem becomes very clear especially when transferring data happens on a high-speed wide area network. Either increasing the TCP window size by tuning network settings or using multiple TCP streams in parallel can be used to overcome this problem and achieve optimal TCP performance. Since lack of automatic network tuning and tuning network settings is different in each every operating system, it cannot be considered as cross platform solution. Hence, we chose multiple parallel TCP streams to achieve maximum bandwidth usage and we will describe in depth about our implementation in this section.

Our idea of multiple parallel TCP streams consists of splitting data into sub small packets at sender side and sending these sub small packets over the network by using multiple Java socket streams in parallel. Although the default socket buffer size is not set to value of the bandwidth delay product, using multiple parallel TCP streams gives better transfer rate by aggregating each socket bandwidth.

Figure 3 illustrates the architecture of NaradaBrokering Parallel TCP (NBPTCP) transport layer, and NBPTCP usage as communication layer between NaradaBrokering *Agent A* and NaradaBrokering *Agent B*. Like all other NaradaBrokering transport protocols, NBPTCP is implemented in the NaradaBrokering's transport layer as multi stream protocol, and it uses our Parallel TCP Socket (PTCPSocket) implementation. PTCPSocket can handle multiple sockets' input and output streams and it is derived from *Java.net.Socket.* It consists of *packet splitter, packet merger, senders, receivers, and TCP sockets*, and it has two types of channels; *communication* and *data channels.* All control information and negotiations are sent over the *communication channel,* which stays open till the end of whole data transfer, and *data channels* are used for actual user data transfer. For example, both sender side and receiver side agree on the number of streams, which will be used during the data transfer by using *communication channel*. Sender side is responsible for deciding the number of parallel streams before initiating the actual user data transfer.

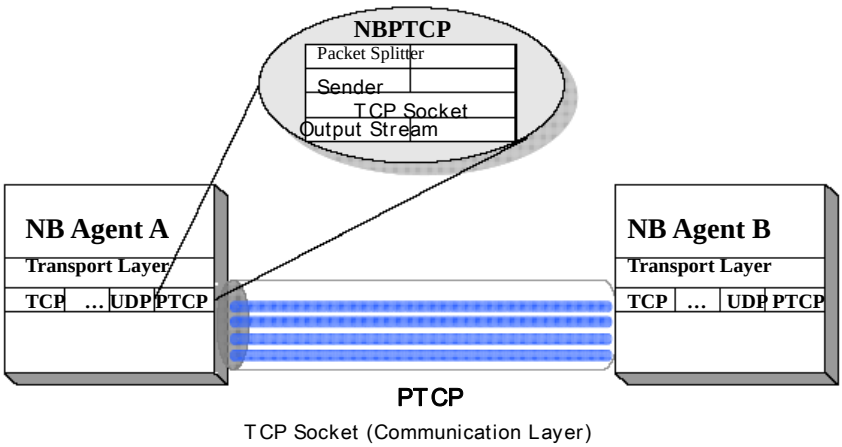

Figure 3 NaradaBrokering PTCP Architecture

After the setting parallel streams' number, *packet splitter* starts diving user data into small packets. These packets are passed to *senders'* layer and *senders* send them to receiver side by writing these packets into *TCP sockets*' output streams (*data channels*). The number of *senders* and *receivers* are same as the number of parallel streams. At receiver side, *receivers* read packets from the *TCP sockets'* input streams (*data channels*) then pass these packets to upper layer, which is called *packet merger*. The *packet merger* combines these incoming packets by checking their packet number, which is given by the *packet splitter*. Since TCP uses a checksum computed over the whole packet to verify that the protocol header and the data in each received packet have not been corrupted, there is no need to check data integrity at the packet merger layer again.

5 Benchmarks

In this section, we will discuss how well our reliable middleware architecture is performing in the existing services. To increase realities, we are done performance tests between Cardiff University at United Kingdom and Indiana University at United State. We are also using multiple platform environments to show interoperability of the NaradaBrokering. For example, we are running NaradaBrokering server on the Windows platform and NB Agents on the Linux platform. The experimental setup is described below (see Figure 1 for each parts):

- GridFTP Client: Dual Pentium III 1GHz CPU with 1.5 GB of RAM on Red Hat Linux 7.2. Located at Cardiff University.
- NB Agent A: Dual Pentium III 1GHz CPU with 1.5 GB of RAM on Red Hat Linux 7.2. Located at Cardiff University.
- NaradaBrokering Server: Pentium 4 2.53GHz CPU with 512 MB of RAM on Windows XP Professional Operating System. Located at Indiana University.
- NB Agent B: Intel(R) Xeon(TM) CPU 2.40GHz CPU with 2GB of RAM on Red Hat Linux 3.2. Located at Indiana University.
- GridFTP Sever: Dual AMD Athlon(tm) MP 1800+ CPU with 513 MB on Red Hat Linux 7.3. Located at Indiana University.

In our performance measurements, we wish to examine the performance penalty represented by adopting the architecture of Figure 1. Again, the routing approach allows us to provide reliability features (such as recovery from network failures) on top of the basic GridFTP file transfer mechanisms. This will create some additional overhead, which we determine below. Note also that the

We will present performance results up to 2 streams since there are virtually no differences beyond 2 streams. This kind of behavior is due to the network setting between Cardiff University at UK and Indiana University at USA, which is beyond our control. Figure 4 shows the performance result of 1 stream of GridFTP, NBGridFTP, and NaradaBrokering. As we can see on this Figure, NBGridFTP is slower by 22.22% (25 MB) to 28.76% (400 MB) range. Those percentages of delays come from inside NaradaBrokering like dividing large file, writing to database, and temporarily copying data on the NaradaBrokering *Agent A* and NaradaBrokering

Agent B. Result of *NB only* represent the performance result of between NaradaBrokering *Agent A* and NaradaBrokering *Agent B*. This means that we remove timing for temporary file store and NaradaBrokering *Agent A* is worked as GridFTP Client and NaradaBrokering *Agent B* is worked as NBGridFTP server. This result gives us idea about how well our NaradaBrokering network implemented. As actual network stand point of view it is only about 11.91% to 18.52% slower compared with GridFTP, plus our NaradaBrokering system has reliable mechanisms are there. As we can see on the Figure 5, we also have similar results for 2 streams case. In this case our architecture is slower compared with GridFTP by 25.44% to 30.91% for *NB + GridFTP* case and about 7.56% to 13.45% for *NB only* case. We also can see the rate of second dropping from the 1 stream case is very similar to GridFTP—GridFTP dropped 42.36% and NaradaBrokering dropped 44.57%. This means our implementation of multiple streams is as effective as what GridFTP has currently. For the future optimization issues, we will discuss about the matters that delays our architecture in the next section.

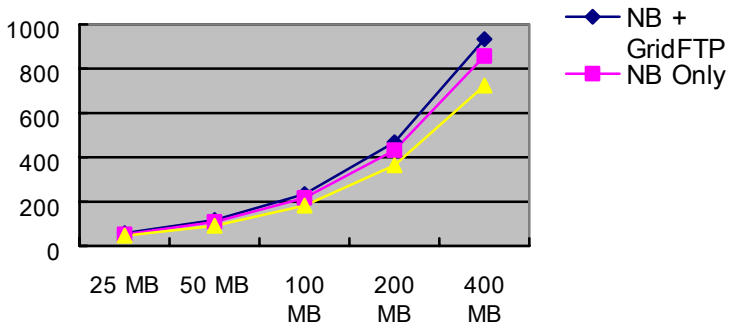

Figure 4 File Transfer Results with 1 Stream

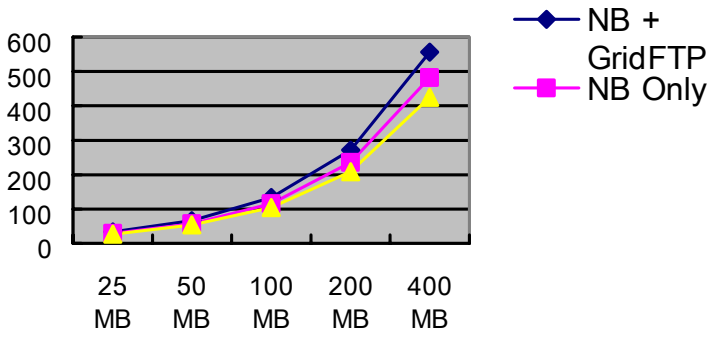

Figure 5 File Transfer Results with 2 Streams

5.1 NaradaBrokering Timing

We will look deeply into the time spent in our architecture for further optimization (see Table 1). We divide NaradaBrokering with GridFTP into 2 parts; Timing for transfer temporary file (from GridFTP client to NaradaBrokering *Agent A* and from NaradaBrokering *Agent B* to GridFTP server) and internal NaradaBrokering time. Internal NaradaBrokering time is divided into initialization, deleting temporary file, writing to database, actual transferring, and merging file. A large file will be divided into small pieces of fixed size and will be stored into temporary directory in the *Initialization* phase and after done transfer, timing for the cleanup those temporary files are measured on the *Delete* phase. Those small pieces of a file will be stored into the database that located on the NaradaBrokering server first. This time is estimated timing based on the experimental benchmark. Actual file transferring time is measured on the *Network* phase. After NaradaBrokering *Agent B* gets all the small pieces of file it will reconstruct original file using those pieces. As we can see for this table, most of the time is either negligible (delete, database, and merging) or non-avoidable (temporary file transfer). And also actual timing for the transferring file is reasonable. According to the Table 2, actual file transfer rates are as good as GridFTP file transfer rates. GridFTP is little bit slower because we did not separate authentication from the actual file transfer.

Table 1. Detailed timing for NaradaBrokering + GridFTP with 2 streams in seconds.

MB	Temporary file transfer	Init	Delete	Database	Merging	Network
25	4.82	0.95	0.02	~ 1	0.36	25.52
50	9.16	1.80	0.05	~ 2	0.72	52.24
100	17.54	3.88	0.11	~ 4	1.66	106.05
200	36.42	17.28	0.22	~ 8	3.15	206.63
400	74.20	41.04	0.43	~ 16	5.97	418.56

Table 2. Timing for actual file transfer only for NB + GridFTP and GridFTP in seconds.

MB	NB + GridFTP transfer	GridFTP Transfer
25	25.52	26.95
50	52.24	54.18
100	106.05	103.93
200	206.63	208.66
400	418.56	424.85

One part we believe we can optimize is initialization. Table 1 shows that it is not taking much time if dealing with small file size. However it takes more then necessary when dealing with larger file size. Initialization phases will be deeply investigated for the future optimization.

6 Conclusions

We discussed reliable transfer mechanism in NaradaBrokering using GridFTP as an example. NaradaBrokering system is an event brokering system designed to run on a large network of cooperating broker nodes and is used here as a general purpose messaging substrate. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. . Decoupling desirable features of existing systems like file recovery technologies in GridFTP from the implementation of the service and instead placing into the reliable, high performance messaging substrate between the client and service will allow us to extend to other services without extensive reimplementation.

We also discussed deploying NaradaBrokering in GridFTP and its performance tests. As we can see from the performance tests we have reasonable file transfer rates with added features like reliable transfer and multiple stream file transfer. We show the possibilities of our goal of decoupling reliability features from the implementation of the service and protocol, and instead placed into the software messaging substrate without great lose of performances.

For future work, the brokering system is by design a many-to-many messaging system, so we may exploit this to support simultaneous delivery of files to multiple endpoints. Finally, we will develop more examples of using other file transfer mechanisms that will mimic RFT-like features without reimplementation

References

[1] GridFTP: Universal Data Transfer for the Grid http://www.globus.org/datagrid/gridftp.html

[2] The Globus Project http://www.globus.org/

[3] Ravi K Madduri, *Reliable File Transfer in Grid Environments*, Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02), 2002.

[4] Reliable File Transfer Service http://www-unix.mcs.anl.gov/~madduri/RFT.html

[5] RFC 765 – File Transfer Protocol specification http://www.faqs.org/rfcs/rfc765.html

[6] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, *GridFTP: ProtocolExtensions to FTP for the Grid*, Argonne National Laboratory, April 2002.

[7] The NaradaBrokering System http://www.naradabrokering.org

[8] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: *A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids.* Proceedings of ACM/IFIP/ USENIX International Middleware Conference. 2003.

[9] G. Fox, S. Lim, S. Pallickara and M. Pierce. *Message-Based Cellular Peer-to-Peer Grids:Foundations for Secure Federation and Autonomic Services.* (To appear) Journal of Future Generation Computer Systems.

[10] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke, *Data Management and Transfer in High Performance Computational Grid Environments* Parallel Computing Journal, Vol. 28 (5), May 2002, pp. 749-771.

[11] G. C. Fox, W. Wu, A. Uyar and H. Bulut *Design and Implementation of Audio/VideoCollaboration System Based on Publish/subscribe Event Middleware* Proceedings of CTS04 San Diego January 2004

[12] S. Lim, G. Fox, S. Pallickara, and M. Pierce, *Web Service Robust GridFTP*. The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA04), June 2004