

GridFTP and Parallel TCP Support in NaradaBrokering

Sang Boem Lim¹, Geoffrey Fox², Ali Kaplan², Shrideep Pallickara² and Marlon Pierce²

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 transfers and parallel transfers are developed as part of the original system. This makes it very hard to re-use the same code for different systems. We address this disadvantage by decoupling useful features of file transfer mechanisms from the implementation of the service and protocol, and instead place these features into the messaging substrate. This will allow us to provide file-transfer capabilities to tools that do not have these features.

1 Introduction

Today's network environments require to people downloading many things daily bases. 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, it has some weaknesses. Since 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 reimplementing 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 bases. 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 by the Globus. 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 *servers* 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. NaradaBrokering imposes a cluster-based structure on the broker network. Clusters comprise strongly connected brokers with multiple links to brokers in other clusters, ensuring alternate communication routes during failures. This distributed cluster architecture allows NaradaBrokering to support large heterogeneous client configurations that scale to a very large size. NaradaBrokering provides support for a wide variety of event driven interactions – from P2P interactions to audio-video conferencing applications.

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 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 middle layer between many different applications, such as Audio/Video [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. They just use NaradaBrokering for communication and acquire reliability through NaradaBrokering.

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).

Support for the Web Service Reliable Messaging Framework (WS-RM) is currently being incorporated into NaradaBrokering. NaradaBrokering is quite resilient to failures since it is based on a distributed broker network and can sustain losses of one or more broker nodes.

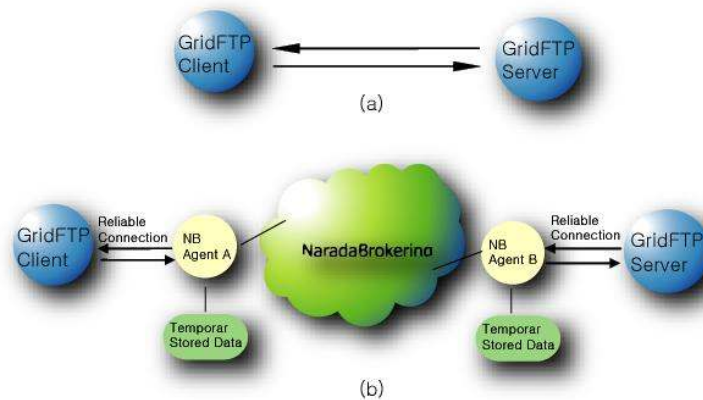


Figure 1 (a) Traditional GridFTP (b) GridFTP with NaradaBrokering

4 Enhancing GridFTP

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

GridFTP and other file transfer mechanisms may already incorporate a number of reliability features on their 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. 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 NaradaBrokering *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 stand 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.

Currently, we have completed development of the uploading functionality of GridFTP with NaradaBrokering 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 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 is occurred 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. 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 ($a_1, a_2 \dots a_{n-1}, a_n$) 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 ant failure cases meanwhile 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 store 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 are occurred 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*.

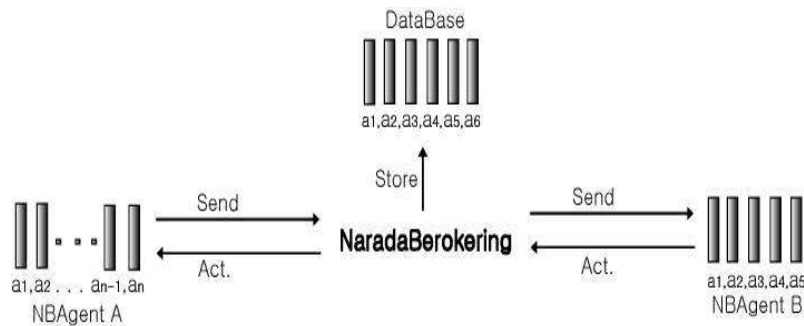


Figure 2 Reliable Mechanisms in NaradaBrokering

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. NaradaBrokering server is smart enough to know resum-ing 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. The main reason why we chose multiple parallel TCP streams to achieve maximum bandwidth usage is that because lack of automatic network tuning and tuning network settings is different in each every operating system, it cannot be considered as cross platform solution 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 them over the network by using multiple Java TCP 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 (PTCP Socket) implementation. PTCP Socket 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*. Overheads and timing are discussed in detail in [13]. Sender side is responsible for deciding the number of parallel streams before initiating the actual user data transfer.

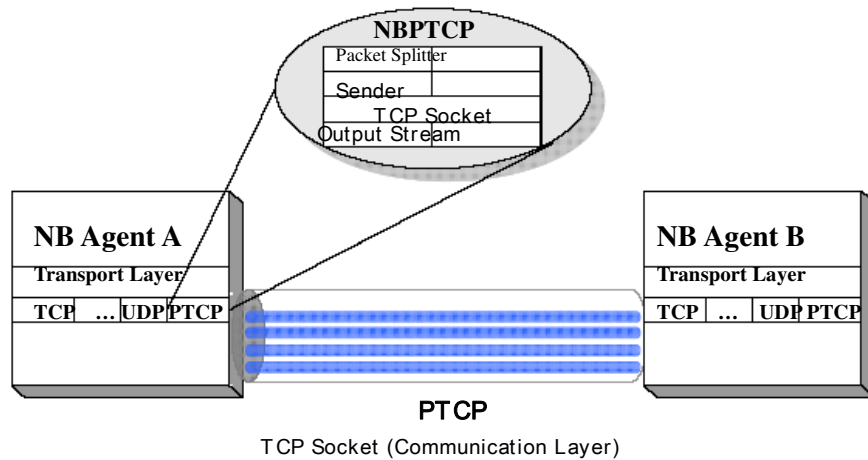


Figure 3 NaradaBrokering PTCP Architecture

After the setting parallel streams' number, *packet splitter* starts dividing 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.

4.2.1 Tests

In this section, we are going to discuss how our multiple stream transfer mechanism architecture is affected by size of the TCP sending and receiving window size and how well it performs. To understand how the underlying network affects its performance, we performed three tests by changing the TCP window size; LAN testing, one continental WAN testing, and one inter-continental WAN testing. To alter TCP window size we used java socket methods `setSendBufferSize(TCP_WINDOW_SIZE)` and `setReceiveBufferSize(TCP_WINDOW_SIZE)`. All the bandwidth capacity was measured by using Iperf with options listed below:

```
iperf -s -w 256k
iperf -c <hostname> -w 512k -P 40
```

4.2.1.1 LAN Test:

It was performed between two Indiana University machines which are nearly 50 miles away from each other. We used the following environments for our performance tests.

Server: Sun Fire V880 machine has 8x1.2 GHz UltraSPARC III processors with 16 GB of RAM on Solaris 9. It has 6x72GB 10K rpm internal HD.

Client: Sun-Fire-V250 machine has 2x1.2 GHz UltraSPARC III processors with 8 GB of RAM on Solaris 9. It has 6x72GB 10K rpm internal HD.

Bandwidth reported by IPerf: 94.6 Mbps

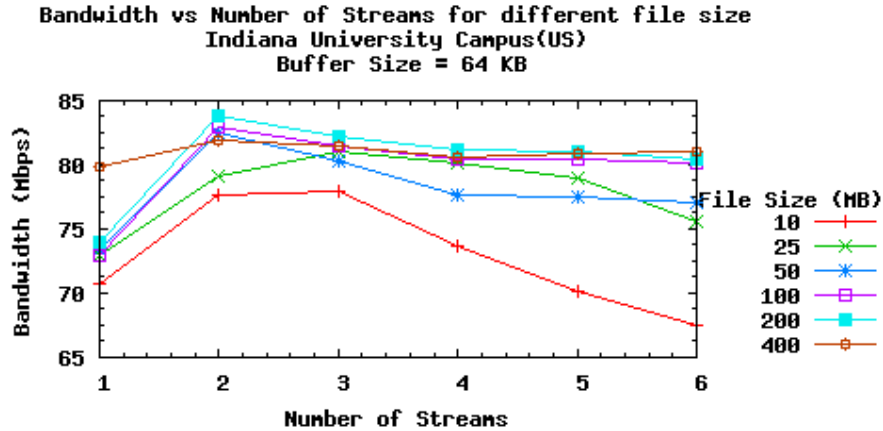


Figure 4: Bandwidth for different file size with a fixed buffer size. (IU-IU settings)

As discussed in **Ref [13]**, even though, there are very fast LAN connections, network transmission time still plays an active role in data transfer over the network. As it can be seen in figure 4, all files get benefits by using multiple parallel streams. However, since transmission time is not as big as WAN, using multiple streams beyond 2 or 3 does not provide any gain in the network bandwidth usage, in fact it degrades the performance. Although, these overheads are less drastic because of longer transmission times associated with them in data transmission of larger size files, smaller size files suffer significantly in the use of multiple streams.

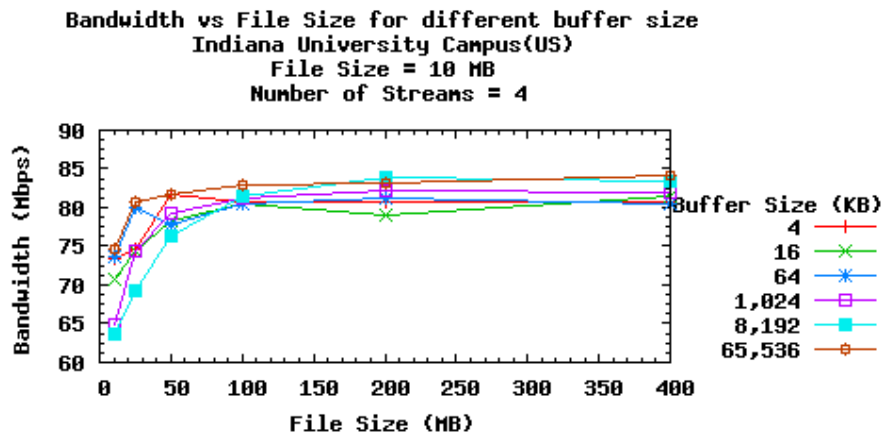


Figure 5: Bandwidth for different buffer size with a fixed stream number. (IU-IU settings)

On the contrary of multiple streams, increasing TCP buffer size has positive impact on LAN based data transmission of all size files. However, for smaller size files, in figure 5, bandwidth usage gains due to increase of buffer size are much better than files sizes are larger than 100MB.

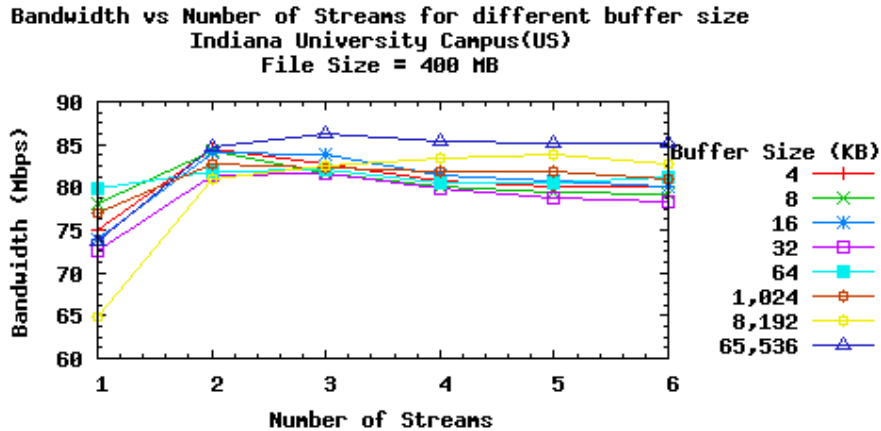


Figure 6: Bandwidth for different buffer size with a fixed file size. (IU-IU settings)

The graph in Figure 6 shows how changing the TCP buffer size impacts network bandwidth usage of file with fixed size (400 MB). It demonstrates that increasing TCP buffer size with 2 or 3 multiple parallel streams boosts network bandwidth usage to its peak value. After that point, the overheads of multiple parallel streams becomes dominant and starts to diminish network bandwidth usage.

4.2.1.2 Continental WAN Test:

This test was performed between Indiana University and University of California at San Diego.

Server: Dual Pentium III 731MHz CPU with 512 MB of RAM on GNU/Linux 2.4.21-4.ELsmp located at University of California at San Diego.

Client: Sun Fire V880 machine has 8x1.2 GHz UltraSPARC III processors with 16 GB of RAM on Solaris 9. It has 6x72GB 10K rpm internal HD.

bandwidth reported by IPerf: 89.4 Mbps

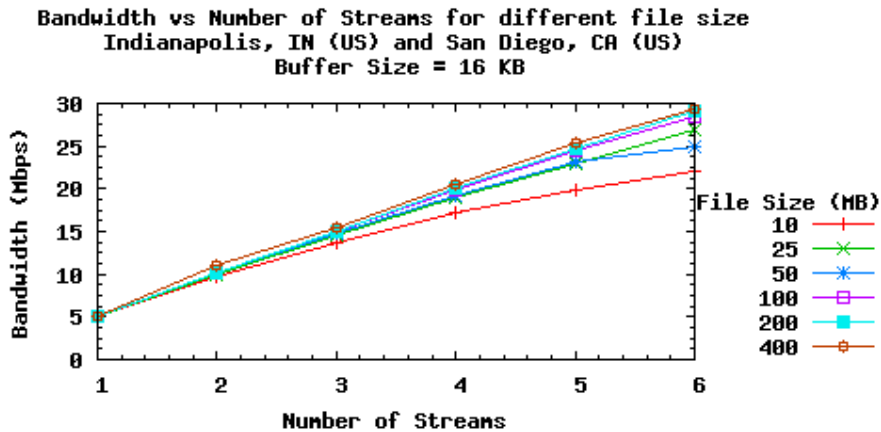


Figure 7: Bandwidth for different file size with a fixed buffer size. (IU-UCSD settings)

As we can see in Figure 7, with a fixed size TCP buffer, the gain from the multiple parallel streams becomes dominant in long-distance data transfer. Although, both smaller and larger size file gets the benefits of multiple streams, the overhead of fragmentation and coalescence of data still has negative impact on files which sizes are smaller than 50 MB.

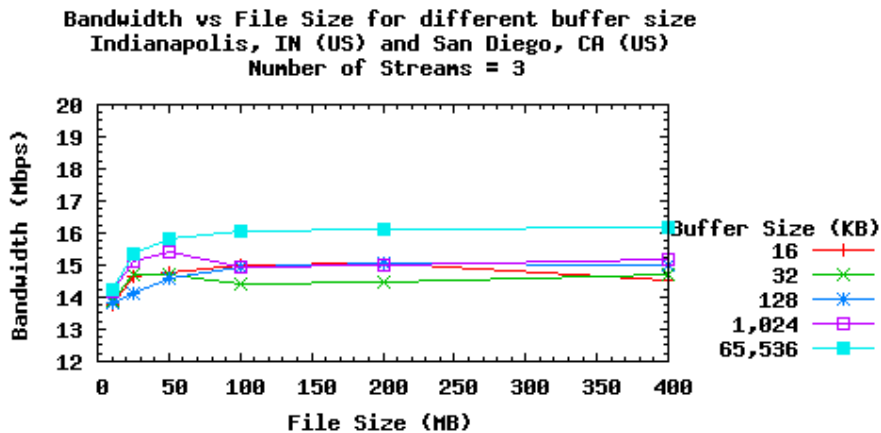


Figure 8: Bandwidth for different buffer size with a fixed stream number. (IU-UCSD settings)

Figure 8 demonstrates that increasing buffer size with a fixed stream number has better positive impact on file size larger than 50 MB. In the case of smaller file sizes the bandwidth usage still suffers from the overhead caused by multiple streams and files smaller than 400 MB size have almost the same bandwidth usage gains despite the significant changes on TCP buffer size. On the other hand, larger size files greater than 400 MB have much better network performance when the TCP buffer size increases considerably.

**Bandwidth vs Number of Streams for different buffer size
Indianapolis, IN (US) and San Diego, CA (US)
File Size = 10 MB**

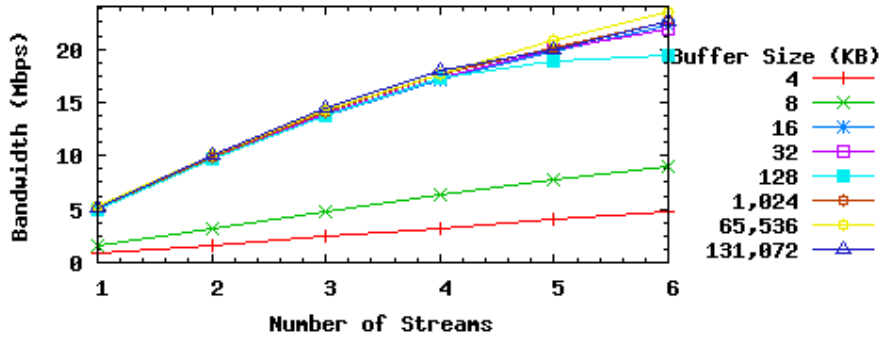


Figure 9: Bandwidth for different buffer size with a fixed file size. (IU-UCSD settings)

As we can see from figure 9, both increasing buffer size and the number of streams boosts the usage of network bandwidth. However, increasing buffer size does not have the same impact as the number of streams. There is no difference in the bandwidth usage when the buffer size reaches 16 KB; however, the bandwidth usage is still rising drastically with the number of parallel streams.

4.2.1.3 Intercontinental WAN Test:

We performed this test between Indiana University at US and XXXXX at Korea. We used the following environments for our performance tests.

Server: Sun Fire V880 machine has 8x1.2 GHz UltraSPARC III processors with 16 GB of RAM on Solaris 9. It has 6x72GB 10K rpm internal HD.

Server(Korea):??????????????

bandwidth reported by IPerf: ???????????? Mbps

**Bandwidth vs Number of Streams for different file size
(US and Korea)
Buffer Size = 32,768 KB**

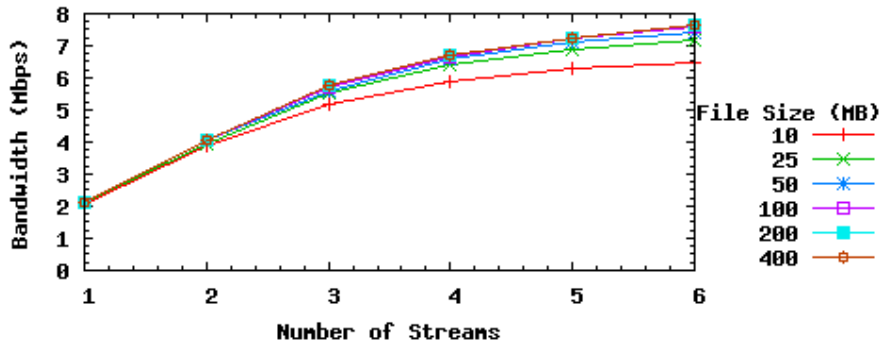


Figure 10: Bandwidth for different file size with a fixed buffer size. (US-Korea settings)

With a fixed size TCP buffer, there is a massive gain from the multiple parallel streams in intercontinental data transfer (in Figure 10). On the contrary of continental WAN data transfer small size files are not suffering the overhead caused by multiple parallel streams.

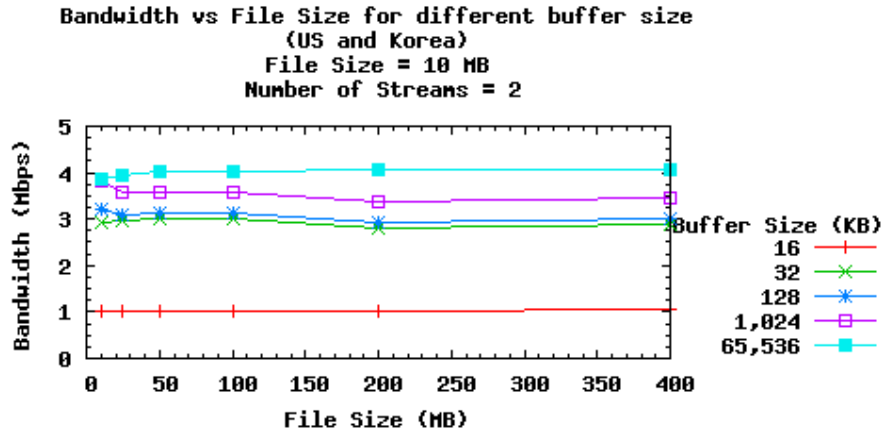


Figure 11: Bandwidth for different buffer size with a fixed stream number. (US-Korea settings)

Figure 11 demonstrates clearly that increasing buffer size with a fixed stream number has vital impact on all size of files. Similar to figure 10, intercontinental data transfer benefits tremendously from increasing of TCP buffer size. Even though, there is an increase in bandwidth usage gain when the buffer size is set to from 1,024 Kb to 65,536 KB, the performance gain is not that much dramatically.

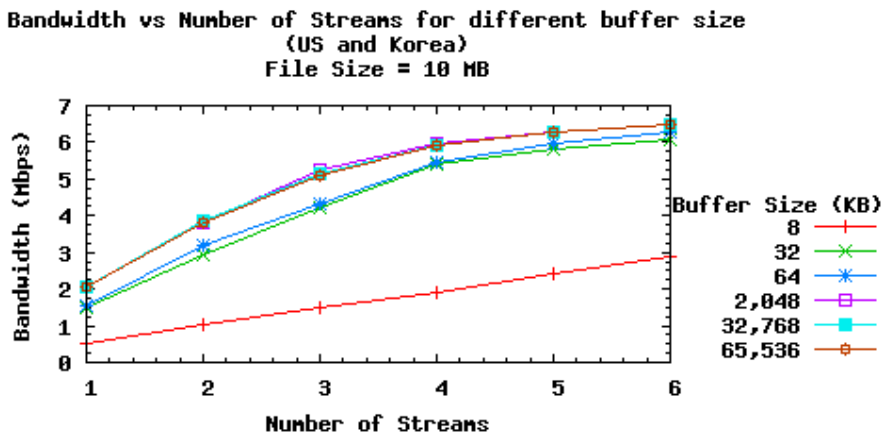


Figure 12: Bandwidth for different buffer size with a fixed file size. (US-Korea settings)

Similar to continental data transfer, as we can from figure 12, the usage of network bandwidth is improved by both increasing buffer size and number of streams. How-

ever, increasing buffer size does not have the same impact as number of streams. There is no difference in the bandwidth usage when the buffer size reaches 2,048 KB; however, the bandwidth usage is still rising drastically with the number of parallel streams.

4.2.2 Future Works

Both LAN and WAN test result show that increasing TCP buffer size and using multiple streams improves network bandwidth usage. However, there is no direct ratio between these. Therefore, finding the optimum buffer size with the most advantageous multiple parallel streams number without reducing the network bandwidth usage and adding extra computation work to underlying system is the major challenging problem that we are planning to solve to provide the best data transmission rate with the minimum system and network overheads.

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.

We are using following environments for our performance tests (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.

We will present performance results up to 2 streams since there are virtually no differences beyond 2 streams. It is happened to both GridFTP and NBGridFTP. 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 are come from inside of NaradaBrokering like divide large file, writing to database, and temporary copy of 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 compare 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 compare with GridFTP about 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 effect as what GridFTP has currently. For the future optimization issues, we will discuss about the matters that delays our architecture in the next section.

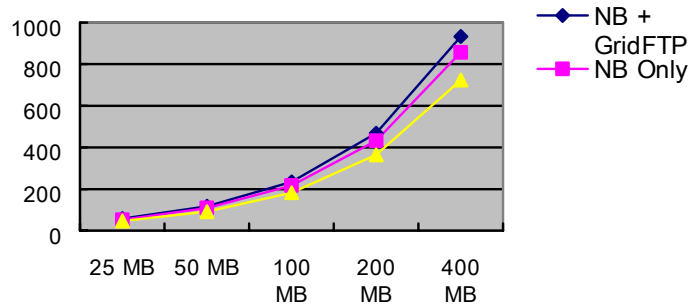


Figure 4 File Transfer Results with 1 Stream

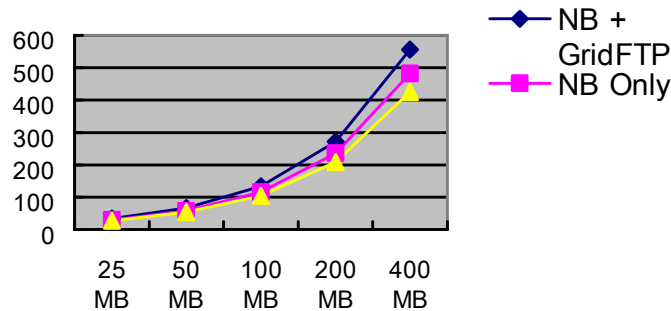


Figure 5 File Transfer Results with 2 Streams

5.1 NaradaBrokering Timing

We will look deeply into the time spending on 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, delete 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 not takes much time (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 form 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 transferring 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 part. Table 1 shows that it is not taking much time if it dealing with small file size. However it takes more then necessary when it is 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. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. . Decoupling good features of exist 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 reimplementaion.

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 great features like reliable transfer and multiple stream file transfer. We show the possibilities of our goal that decouple reliability features from the implementation of the service and protocol, and instead placed into the 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 RTF-like features without reimplementaion

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: Protocol Extensions 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
- [13] Pete Burnap et al. *Worldwide Messaging Support for High Performance Real-time Collaboration*. Proceedings of the UK e-Science Programme's All Hands Meeting 2005 (AHM2005). Nottingham, UK.