GridTorrent Framework: A High-performance Data Transfer and Data Sharing Framework for Scientific Computing

Ali Kaplan[1, 2], Geoffrey C. Fox[1, 2, 3], Gregor von Laszewski[4]
[1]A Community Grids Lab, Indiana University
[2]Computer Science Department, School of Informatics, Indiana University
[3]Physics Department, Collage of Art and Sciences, Indiana University
[4]Center for Advancing the Study of Cyberinfrastructure,
Rochester Institute of Technology
{alikapla,gcf}@indiana.edu, gregor@rit.edu

Abstract

Large amount of data that is often stored in many thousands of files is created as part of today's geographically distributed scientific computation and collaboration environments.

Managing and transferring large volumes of data sets present a significant challenge and are often a bottleneck in the scientific computing community. In this paper, we present an architecture to manage data distributions in a collaborative fashion through a GridTorrent Framework (GTF) whose data transfer mechanism inspired by Bittorrent. We present performance experiment data that compares our framework to parallel TCP (PTCP) and Bittorrent. Experimental results conducted suggest that using GridTorrent for large data set has significant advantages over parallel TCP in LAN and WAN type of computer networks.

Keywords: Grid, data transfer, collaboration, Bittorrent,

1. Introduction

Today's computational science is in many cases based on data-intensive applications. New scientific devices and large-scale observatories generate massive volumes of data sets. The Internet and computational Grid [10, 12] make the data accessible to anyone anywhere by mechanisms of replication, creation, and recreation of more data [13]. Prime examples of domain specific scientific disciplines with these characteristics include high-energy physics and bioinformatics. To illustrate, every year, the Large Hadron Collider (LHC) experiment at CERN generates petabytes of data that is required to be distributed world-wide. Scientists geographically dispersed are interested in analyzing these data sets. Consequently, there is a growing need for efficient techniques to disseminate the data through a simple framework which is integrated into the collaboration environment for the scattered users.

A number of related activities exist in this area of efficient data transfer. GridFTP [11] is the one of the most common data transfer services supported by the Grid community and a key feature of Data Grids [14]. However, due to petascale information, datacenters go beyond the concept of supercomputers with just CPU farms. They must more often include IO and networking arrays [15]. Consequently, a new data transfer technique that has the ability to utilize the available system resources effectively and efficiently is urgently required for many state-of-the-art scientific computations, particularly those needing to integrate Grid resources.

In spite of the fact that peer-to-peer (P2P) systems have the potential of cost sharing and balancing of network resources, there is a lot of research being conducted in this area to investigate advantages and disadvantages of P2P systems [7, 16, 17, 18]. However, many of those studies [16, 17, 18] have shown that scientific community could benefit by exploiting P2P, even though it does not fulfill all the requirements of data transfer for the scientific computing environment, such as security, content access control, and collaborative environment. In addition, Bittorrent has outperformed GridFTP in network areas where only limited bandwidth is available [18].

In this paper, we investigate if a P2P system, i.e. GridTorrent, could be used for the dissemination of large amount of data generated in scientific applications while comparing data transfer performance of our GridTorrent Framework with that of parallel TCP. The distinguishing factors for our GridTorrent are the integration of security features and the conversion of its tracker to Web Service Tracker (WS-Tracker) (See Section 3.1.3 for more details about the tracker).

The remainder of this paper is organized as follows. In the next section, we present the architecture of parallel TCP followed by Section 3 the GridTorrent Framework architecture. Section 4 presents performance benchmarks conducted in two different types of networks (LAN and WAN) and analyze their results. Finally, we conclude and indicate opportunities for future research in section 5.

2. Multiple Stream Transfer Mechanism

It is a well-known fact that TCP's window based congestion control mechanism prevents [1] full-scale usage of high bandwidth-delay product. Hence, transferring large data set across high-performance networks is suffering from limitations of the current TCP implementation [1, 2] as it prevents the use of maximum bandwidth. A solution at the application level is provided by parallel TCP implementations [2, 3]. In the next section, we are going to present briefly the architecture of our Java-based implemented PTCP [4, 5] data transfer mechanism.

2.1. PTCP Architecture

A Parallel TCP stream consists of three basic steps; splitting of data into sub packets at the sender side, sending these sub packets over the network by using multiple streams in parallel, and coalescing of received sub packets at the receiver side. Using multiple parallel TCP streams gives high throughput by aggregating each socket bandwidth, although the default socket buffer size is not set to value of the bandwidth-delay product.

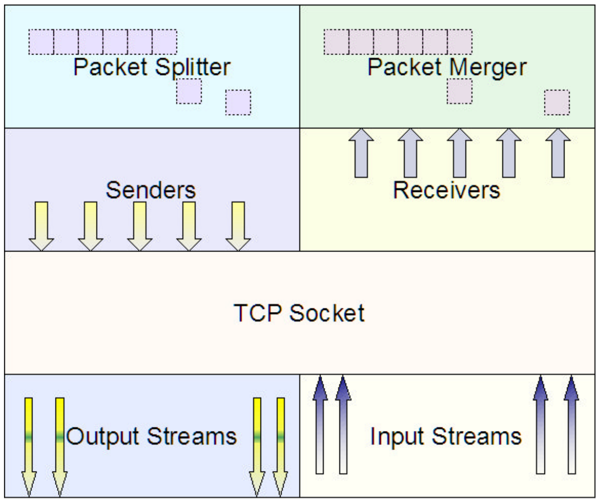

Figure 1: A parallel TCP socket architecture.

Figure 1 depicts the architecture of the Java-based PTCP framework. PTCPSocket derived from *Java.net.Socket* can handle multiple sockets' input and output streams. It is comprised of *packet splitter, packet merger, senders, receivers, and TCP sockets*, and has two type of channels; *communication* and *data channels*. All control information and negotiations are sent over the *communication channel, which* stays open until the end of entire data transfer, and actual data are sent over the *data channels*. For instance, the decision of how many parallel streams are used is determined by the sender and is communicated with the receiver before initiating the actual data transfer through the communication channel.

After the setting the number of parallel streams, the *packet splitter* divides user's data into smaller sub packets. These sub packets are then passed on by the *senders* to the receivers while writing these packets into data channels utilizing *TCP sockets*. The number of *senders* and *receivers* has to be same as the number of parallel streams. R*eceivers* read packets from the *data channels* and pass them to the upper layer *packet merger* at the receiver's side. Merging smaller sub packets into one packet is conducted by the *packet merger*. It combines the incoming packets by checking their packet number assigned by the *packet splitter*. There is no need to check data integrity at the packet merger layer again, 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.

3. GridTorrent Data Transferring and Sharing Framework

Parallel TCP could address the low performance data transfer [4, 5, 6] problem caused by TCP's window based congestion control mechanism, enabling aggregation of data transfer throughput by using multiple data stream sockets from the same source to same destination. However, when there are numerous requests for a single data source, that source becomes a bottleneck. To alleviate this problem, peer-to-peer data distribution strategies can be exploited effectively. There are three broad categories of techniques [7] to optimize large data distribution at application level: data staging, data partitioning, and exploiting orthogonal bandwidth in peer-to peer data dissemination.

We chose the Bittorrent [8] protocol because it supports data partitioning and orthogonal bandwidth exploitation features [7, 8]. It uses tit-for-tat [8] scheme to enforce participation and fair sharing. Hence, implementing a data transfer framework based on Bittorrent enables having multiple streams to multiple destinations instead of having multiple streams from one source to many destinations. This approach alleviates the bandwidth bottleneck nodes and load balancing in the overall system.

However, there are several requirements set by the scientific community that do not yet met by Bittorrent and address distributing and managing scientific community data.

The first reason is the nature of data. The data in Bittorrent community is generally obtained from other people's works, like music or movie file, whereas every scientific data is generated in scientific community. Therefore, scientific data are more sensitive than data used in Bittorrent community.

The second reason is base on the users' characteristics. In regular Bittorrent community, there is no competition between users. There is one type of user, a passive user, and any user can access any data as long as he or she gets the torrent file. However, in the scientific community, due to expertise or research agenda and competition between institutions, only authorized users are permitted to access to the pre-determined data sets with some access rights. While the passive user type in Bittorrent, the users in scientific community area are very active and some of them cooperate on some files as a group. This creates diverse profiles of users and groups in scientific community.

Third reason is based on the importance of data and its access. Since, the current Bittorrent design itself does not provide a search facility to find files by name; a user must find the initial torrent file by other means, such as a web search [8]. On the other hand, searching, finding, and accessing to desired data are of paramount importance in scientific community, hence a reliable search service must be offered to users working in scientific community.

Therefore, there is a need for integration of Bittorrent and content and collaboration framework with a search facility to use Bittorrent in scientific community. However, our main objective is to compare parallel TCP's experiment results with GridTorrent's to observe whether using GridTorrent for scientific data transfer could offer acceptable data transfer rate besides its efficient system's resource usage. Hence, in the next sections, we are going to describe its architecture and components in brief. Also note that, since parallel TCP does not have security feature, in GridTorrent tests, to avoid biased test results, we did not involve any processes with regard to security, either.

3.1. GridTorrent Framework Architecture

GridTorrent Framework (GTF) has three major components: the GridTorrent Framework Client (GTFC), the WS-Tracker, and the Collaboration and Content Manager (CCM).

Figure 2 presents the basic architecture of interactions between GridTorrent components. Users are the people who interact with the system through CCM. GTFC is software that runs on users' computers and communicates with GTF by exchanging SOAP messages with WS-Tracker.

The process is started by users (human being) by registering to Collaboration and Content Manager. Then, they publish their content to by selecting different access level of it. Afterwards, this information will be delivered to users' GridTorrent client via WS-Tracker. After receiving task list, i.e. mentioned information and delivered by WS-Tracker, firstly, GridTorrent Client starts to building .torrent file. Secondly, it announces it to WS-Tracker. Finally, it waits other peers to deliver content of shared file.

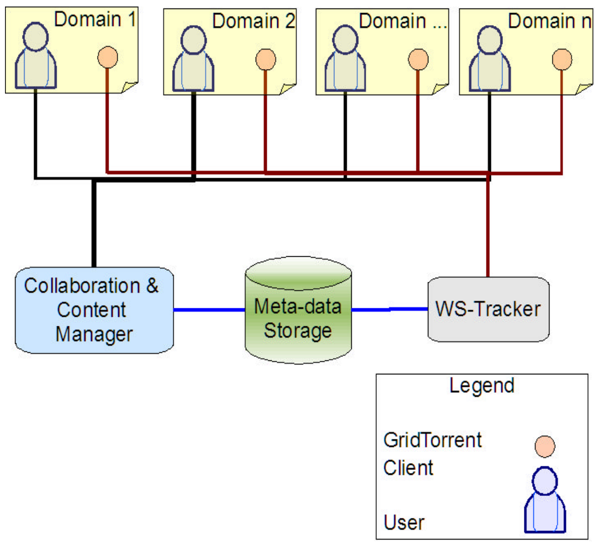

Figure 2: Interactions between GridTorrent Framework components.

3.1.1. Collaboration and Content Manager

As the name indicates, the Collaboration and Content Manager (CCM) has two subcomponents; content manager and collaboration manager.

The *Content Manager* allows users to publish or share their files with selected access control rights. Three types of access level are supported; public, group, and user level accesses. Each content file must be assigned an access level. A content in public access level is accessible publically to all users. A content with group access level permits users to share their contents with selected group members. A group membership process is started either by the group owner or by the user making a request for a desired group. In both cases, group membership is activated after acceptance of both sides. However, since resignation of a group membership does not require both sides' endorsements, it is a one-sided activity. Either the group owner or the user can revoke it. In user level access, content owners can choose individual users by name either from a list of all known users or a self-maintained Buddy list. Users can search for content through Content Subscriber component.

The *Collaboration Manager* permits users to build a virtual sharing environment by managing working groups or friend list. Collaboration tools are composed of Group and Buddy Management subcomponents as illustrated in Figure 3.

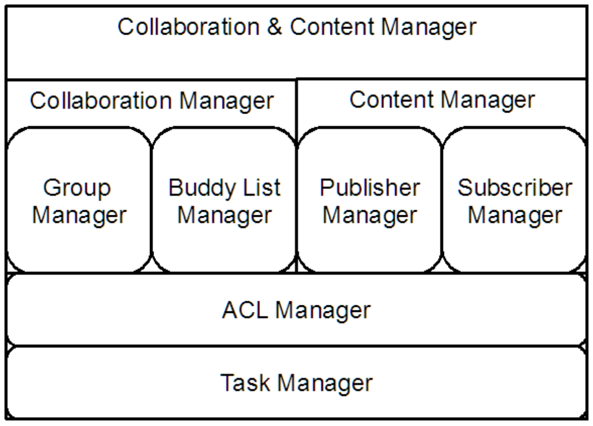

Figure 3: Components of Collaboration and Content Manager.

3.1.2. GridTorrent Framework Client

The GridTorrent Framework Client is responsible for initiating actual data publishing, data sharing with other GTF clients and ensuring secure environment for the aforementioned activities. Figure 4 illustrates the architecture of GTF client.

The GridTorrent Framework client is composed of five main components: Bittorrent data sharing algorithm, task manager, WS-Tracker client, sockets, and security manager. Bittorrent data sharing algorithm uses Bittorrent algorithm to exchanges data between other GTF clients in peer-to-peer manner. Task manager make sure the tasks in the user's task list entered to system by using Collaboration and Content manager and received from WS-Tracker will be performed. WS-Tracker client behaves as a communication layer between task manager and WS-Tracker. Sockets module is responsible for sending and receiving actual data from other peers. Security manager handles issues related to security, for instance exchanging certificates, encrypting and decrypting of messages.

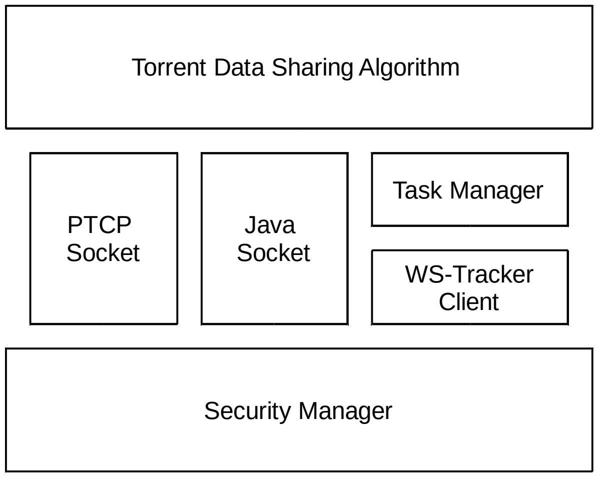

Figure 4: GridTorrent Framework Client Architecture

3.1.3. WS-Tracker

WS-Tracker is a server which assists in the communication between peers using the Bittorrent protocol. Even though, WS-Tracker seem similar to Bittorrent tracker in this aspect, there are quite differences between them in regard to WS-Tracker's functionalities. In Bittorrent, tracker only delivers list of available seeders and peers of a requested files, and collects statistics of uploading and downloading processes. After the initial communication, peer can continue without a tracker. However, in GridTorrent Framework, it acts as a maestro between real users and their GridTorrent clients and other GridTorrent clients. Task lists generated by users are delivered to GridTorrent clients through WS-Tracker. Additionally, access control list of each shared file is supplied to GridTorrent clients by WS-Tracker.

4. Experimental Results

In this section, we will discuss how well our GridTorrent Framework's data transfer mechanism architecture is performing. To observe how the underlying networks influence its performance, we have set three scenarios and conducted two tests: LAN testing, continental WAN testing. Table 1 shows technical features of machines used in different locations.

In each scenario, to compare PTCP's and GridTorrent's performance, we used both PTCP and GridTorrent test cases. We chose 300 MB for file size because the study [9] has shown that the mean file size generated in scientific computation community is larger than 300MB in.

To measure the practical maximum available bandwidth capacity of the underlying network, we used Iperf, a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss. To assess the maximum TCP bandwidth, we tried several TCP window size along with the parallel stream number. In LAN and WAN tests, TCP window size was set to maximum value allowed by the underlying operating system. Note that since the operating systems are not same on test machines, TCP window size used for LAN and WAN are not the same.

Name	**Specifications**	**Network**	**Institution**	**Location**
A	Intel(R) Quad-Core Xeon(TM) 4x2.33GHz CPU with 8GB of RAM	Broadcom NetXtreme II BCM5708 1000Base-T	Indiana University	Bloomington, IN

	on Red Hat Enterprise Linux 4.0			
B	Sun Fire V880 8x1.2GHz UltraSPARC III processors with 16GB of RAM on Solaris 9. It has 6x72GB 10K rpm internal HD	Gigabit Ethernet and 10/100-BaseT Ethernet	Indiana University	Indianapolis, IN
C	Dual Pentium III 731MHz CPU with 512MB of RAM on GNU/Linux 2.6.20-1.2316.fc5	Gigabit Ethernet and 10/100-BaseT Ethernet	Florida State University	Tallahassee, FL

Table 1: Server and client machines' descriptions and their locations

4.1. LAN Test

It was performed between two Indiana University's machines nearly 50 miles apart. Theoretical available bandwidth capacity is the maximum data transfer rate which the underlying network interface card allows. Measured available bandwidth capacity is assessed by using Iperf with the following parameters.

Theoretical Available Bandwidth: 1000 Mbps
Measured Available Bandwidth: 857 Mbps

Server side: Iperf -s -w 256k
Client side: Iperf -c <hostname> -w 512k -P 50

4.1.1 Scenario I

The purpose of this scenario is to observe the performance of PTCP and GridTorrent in local area network. Therefore, server and client machines are in local area network. For the performance test of PTCP, we used one client and one server. The number of parallel TCP stream between server and client has risen from one to sixteen in increment of one stream in each step. Figure 5 demonstrates the connections diagram of PTCP test case.

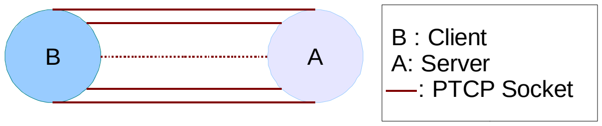

Figure 5: Client and server configuration for PTCP test case. Server is located at Bloomington, IN, whereas client is at Indianapolis, IN.

The connections topology between GridTorent client and seeders are displayed in Figure 6. In GridTorrent test case, one Java socket has been used between each seeders and the peer. The test has been initiated into one seeder and the number of seeders was raised by one in each step, up to sixteen.

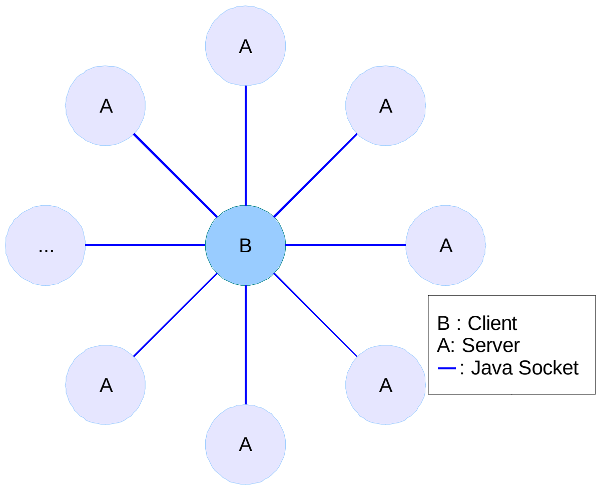

Figure 6: GridTorrent test case configuration for LAN test. Regular Java sockets are used for data transfer.

4.1.2. LAN Test Result

In LAN test, there is no significant improvement in bandwidth usage while using multiple parallel streams [4, 5] because of today's very fast LAN connection. Furthermore, transmission time is smaller than overhead time in LAN; thus, any overhead process substantiality deteriorates data transfer rate, since the experimental data transfer (80-100 Mbps) rate is much lower the theoretical (1000Mbps) and the measured data transfer rate (857Mbps). As it is seen from Figure 7, the deterioration does not have identifiable pattern when the number of parallel streams is increased. The network instability might cause that random fluctuation.

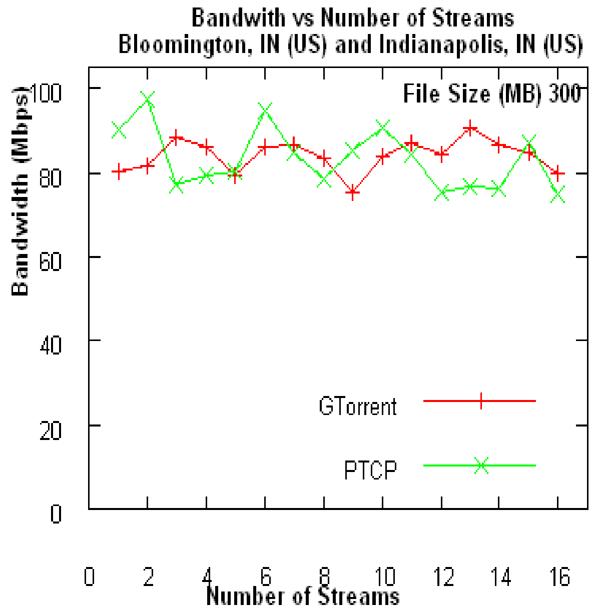

Figure 7: Bandwidth for different stream numbers with a fixed file size. (IU-IU settings)

4.2. Continental WAN Test

This test was performed between Indiana University at Bloomington and Florida State University at Tallahassee, and two scenarios were tested.

Theoretical Available Bandwidth: 1000 Mbps
Measured Available Bandwidth: 30.2 Mbps

Instead of 512kB buffer size used in LAN test; we set TCP window buffer size to 256kB because of underlying network characteristic. We used Iperf with the following options to measure the maximum available bandwidth capacity of the underlying network,

For server side: Iperf -s -w 256k
For client side: Iperf -c <hostname> -w 256k -P 50

4.2.1. Scenario II: GridTorrent Framework Client with One Socket

This scenario is very similar to scenario I, except the location of client located at Florida State University in Tallahassee, FL. In scenario, one client and one server have been used for PTCP performance test, and the number of parallel TCP stream was increased from one to sixteen streams in increment of one stream. Figure 8 illustrates the connections diagram of PTCP test case.

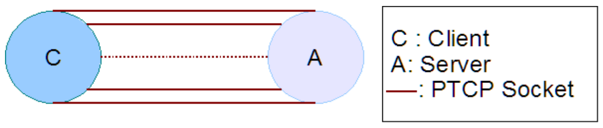

Figure 8: Client and server layout for PTCP test case. Parallel TCP streams were used for data transfer. Server is located in Bloomington, IN, whereas client is in Tallahassee, FL.

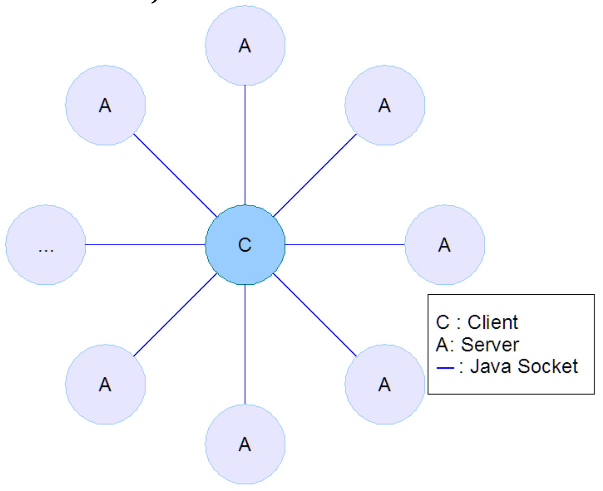

Figure 9: GridTorrent test case topology for wide area network test. Regular Java sockets are used for data transfer.

4.2.2. Scenario II: Test Result

The gains in terms of accomplished data transfer rate are substantial, when the multiple parallel streams used in long-distance to transfer data. Test results were agreed with the above premise. As seen in Figure 10, bandwidth usage is vastly improved in both GTF and PTCP.

PTCP's bandwidth utilization rate has risen steadily until fifteen streams. It has its peak value of 118 Mbps. Just after the fifteenth stream, its data transfer rate starts falling.

GTF has displayed the same characteristic; instead of fifteenth stream, its bandwidth usage rate begins to decline right after thirteenth streams. GridTorrent was performing better than PTCP when the number of parallel streams is less than five. Between the fifth and thirteenth streams, it demonstrates that it has slightly better data transfer rate than PTCP's. Another interesting outcome is that the maximum achieved data transfer rate we measured is almost four times higher than Iperf 's result because Iperf is used as a standard network bandwidth measurement tool among computer users. Another advantage of GTF is feature of load balancing. Whereas the whole data is sent from a single source in PTCP setup, approximately 1/(number of seeds) of the whole data is sent from a single seed in GTF setup. This feature will help to relieve the bottleneck problem of a single source under a great many requests of data transfer.

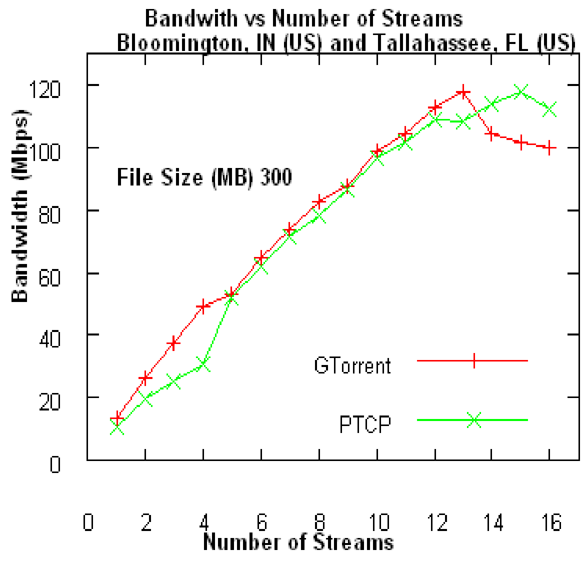

Figure 10: Bandwidth for different stream numbers with a fixed file size. (IU-FSU settings). GridTorrent uses one socket in each connection for every source.

4.2.3. Scenario II: GridTorrent Framework Client with Four Sockets

Besides Java socket, other data transfer protocols can be exploited in GridTorrent client. In order to investigate the performance of the combination of multiple parallel TCP streams and Bittorrent algorithm in wide area network, in this scenario, ,instead of one Java socket, as it is seen in Figure 12, four parallel TCP sockets were used between peer and seeders, and number of seeders has commenced from one and increased from one to 16. It was raised by one in each step.

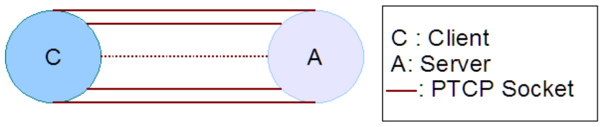

Figure 11: Client and server layout for PTCP test case. PTCP streams are used for data transfer.

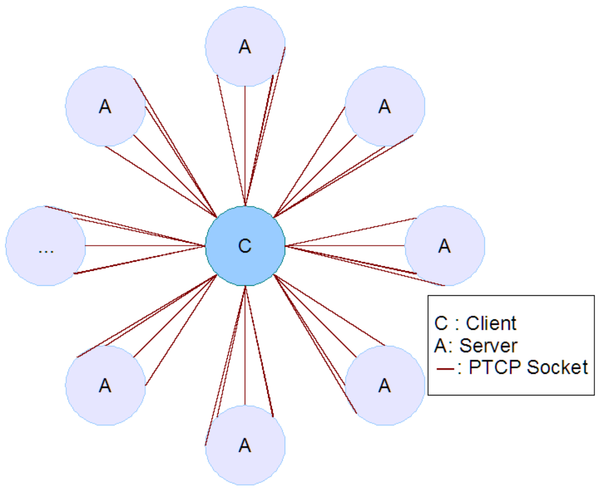

Figure 12: GridTorrent test case topology for WAN test. Four parallel TCP sockets are used for data transfer.

4.2.4. Scenario III: Test Result

Parallel TCP test topology in Figure 11 was same and conducted exactly as in previous scenario. The results were very encouraging using parallel TCP with Bittorrent algorithm by demonstrating much better bandwidth usage than standalone GridTorrent and PTCP. The maximum attained bandwidth is around 145 Mbps which is %23 higher than PTCP's result (118 Mbps). Figure 13 presents substantial increment in data transfer rate when multiple parallel streams are used in GridTorrent. This result is important, because increments in parallel streams of PTCP does not provide any gain after 15 streams; in fact, it deteriorates the data transfer rate. However, we could increase the number of parallel streams in GTF up to 40 (10 seeds) while without any decrease in the data transfer rate.

We also set number of parallel stream and seeders to different values to obtain the maximum achievable bandwidth. 160 Mbps is the maximum accomplished bandwidth by using five parallel multiple streams with eight seeders.

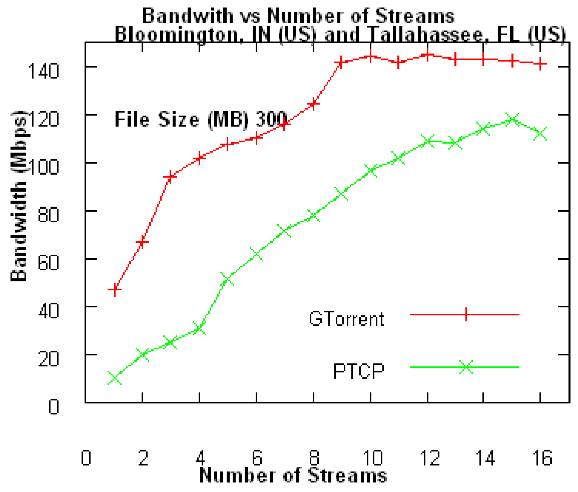

Figure 13: Bandwidth of different stream numbers for a fixed file size. (IU-FSU settings). GridTorrent client uses four parallel TCP sockets in each connection for every source.

4.3. Overhead

Both parallel TCP and GridTorrent have overhead due to nature of multiple parallel connections. Since data splitting and coalescing take place for the entire data transfer, they are the common overhead processes in both PTCP and GridTorrent. In addition to that, GridTorrent has to fragment the file into chuck when the desired file is shared. It is a one-time process, in contrast to data splitting and merging processes.

PTCP's communication channel overhead time can be compared to GridTorrent WS-Tracker client's overhead time which is varying from 300 to 600 milliseconds. Another overhead of GridTorrent is that control messages exchanged between peers to ensure Bittorrent protocol rules strictly enforced to all participating peers. Our testing results demonstrated that the total size of overhead messages is between 148KB to 169 KB. This overhead can be ignored when it is compared to file size of 300 MB

5. Conclusion and Future Work

The objective of GridTorrent Framework is to provide an application level data transferring and sharing framework for data intensive applications over high performance networks such as scientific computing. Due to its P2P nature and Bittorrent protocol, it provides a data transfer technique, which has ability to efficiently utilize the available system resources, such as network bandwidth, IO and CPU. Additionally, the experiment results have shown the performance of GTF is better or not worse than that of parallel TCP. This outcome is important since parallel streaming is used in many scientific computing data transfer tools such as GridFTP. Using Java socket and parallel TCP indicates that, GTF can exploit other high performance data transfer protocols like GridFTP or UDT in traditional low BDP environments at the same time.

GridTorrent Framework has been implemented and released as an open source project. In our future work, the integration of GTF with other high performance low level data transfer protocol mentioned above, and their performance over different network structure will be investigated.

6. References

[1] D. Katabi, M. Hardley, and C. Rohrs, "Internet Congestion Control for Future High Bandwidth-Delay Product Environments," presented at ACM Sigcomm 2002 Conference, Pittsburgh, PA, USA, August 2002.

[2] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in *ACM SIGCOMM 2000*, 2000, pp. 43-56.

[3] Y. Zhang, E. Yan, and S. K. Dao, "A Measurement of TCP over Long-Delay Network," in *Proceedings of the Sixth International Conference on Telecommunication Systems, Modeling and Analysis*, 1998, pp. 498-504

[4] S. Lim, G. Fox, A. Kaplan, S. Pallickara and M. Pierce "GridFTP and Parallel TCP Support in NaradaBrokering," in *Proceedings ICA3PP 2005 of 6th International Conference on Algorithms and Architectures for Parallel Processing*, Melbourne Australia. October 2-5 2005. Springer-Verlag Lecture Notes in Computer Science, Volume 3719, pp. 93-102 DOI

[5] P. Burnap, H. Bulut, S. Pallickara, G. Fox, D. Walker, A. Kaplan, B. Yildiz, and M. A. Nacar "Worldwide Messaging Support for High Performance Real-time Collaboration" presented at Proceedings of the UK e-Science All Hands Meeting 2005, Nottingham, UK, September, 2005.

[6] Sivakumar, H, S. Bailey, R. L. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks", presented at Proceedings of IEEE Supercomputing 2000, November 2000. http://www.ncdm.uic.edu/html/psockets.html

[7] S.A. Kiswany, M. Ripeanu, A. Iamnitchi, S. Vazhkudai, "Are Peer-to-Peer Data Dissemination Techniques Viable in Today's Data Intensive Scientific Collaborations?," in *Proceedings of the 13th International Euro-Par Conference: European Conference on Parallel and Distributed Computing*, Rennes, France, August 2007

[8] Cohen, B., BitTorrent web site: *http://www.bittorrent.org/index.html.* 2005

[9] Iamnitchi, A., Doraimani, S., and Garzoglio, G. "Filecules in High-Energy Physics: Characteristics and Impact on Resource Management" in 15th IEEE International Symposium on High Performance Distributed Computing, 2006. Paris, France, June 2006, pp. 69-80

[10] I. Foster, C. Kesselman, and S. Tuecke "The Anatomy of the Grid:Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, Volume 15, No. 3, pp. 200-222, 2001.

[11] W. Allcock , J. Bester , J. Bresnahan , A. L. Chervenak , C. Kesselman , S. Meder , V. Nefedova , D. Quesnel , S. Tuecke , I. Foster, "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing," in *Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, April 2001, p. 13.

[12] I. Foster and C. Kesselman, *The Grid: Blueprint for a new Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 1999.

[13] Graham, S.L. Snir, M., Patterson, C.A., (eds), Getting Up To Speed, The Future of Supercomputing, NAE Press, 2004, ISBN 0-309-09502-6

[14] The Globus Project http://www.globus.org/ 2007

[15] G. Bell, J. Gray, A. Szalay "Petascale Computational Systems," *Computer, IEEE*, Volume 39, Issue 1, pp. 110-112, January 2006, ISSN 0018-9162

[16] B. Wei, G. Fedak, F. Cappello, "Collaborative Data Distribution with BitTorrent for Computational Desktop Grids," in *Proceedings of the 4th International Symposium on Parallel and Distributed Computing*, July 2005, pp. 250 – 257.

[17] B. Wei, G. Fedak, F. Cappello, "Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent", in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, November 2005, p. 8.

[18] A. Zissimos, K. Doka, A. Chazapis, and N. Koziris. "GridTorrent: Optimizing data transfers in the Grid with collaborative sharing," presented at 11th Panhellenic Conference on Informatics (PCI2007), Patras, Greece, May 2007.