Big Data Transfers Using Recursive Chunk Division

[1] Prof Nilima Nikam, [2] Ajinkya Sarvankar

[12] Yadavrao Tasgaonkar Institute Of Engineering And Technology Dept. Of Computer Engineering

Abstract - *In end-to-end data transfers, there are several factors affecting the data transfer throughput, such as the network characteristics (e.g. network bandwidth, round-trip-time, background traffic); end-system characteristics (e.g. NIC capacity, number of CPU cores and their clock rate, number of disk drives and their I/O rate); and the dataset characteristics (e.g. average file size, dataset size, file size distribution). Optimization of big data transfers over inter-cloud and intra-cloud networks is a challenging task that requires joint-consideration of all of these parameters. This optimization task becomes even more challenging when transferring datasets comprised of heterogeneous file sizes (i.e. large files and small files mixed). Previous work in this area only focuses on the end-system and network characteristics however does not provide models regarding the dataset characteristics. In this study, we analyze the effects of the three most important transfer parameters that are used to enhance data transfer throughput: pipelining, parallelism and concurrency. We provide models and guidelines to set the best values for these parameters and present two different transfer optimization algorithms that use the models developed. The tests conducted over high-speed networking and cloud testbeds show that our algorithms outperform the most popular data transfer tools like Globus Online and UDT in majority of the cases.*

Key Words: GridFTP, FTP, Recursive Chunk Division(RCD), FutureGrid

1. INTRODUCTION

1.1 Existing System

Most scientific cloud applications require movement of large datasets either inside a data center, or between multiple data centers. Transferring large datasets especially with heterogeneous file sizes (i.e. many small and large files together) causes inefficient utilization of the available network bandwidth. Small file transfers may cause the underlying transfer protocol not reaching the full network utilization due to short-duration transfers and connection start up/tear down overhead; and large file transfers may suffer from protocol inefficiency and end-system limitations. Application-level TCP tuning parameters such as pipelining, parallelism and concurrency are very affective in removing these bottlenecks, especially when used together and in correct combinations. However, predicting the best combination of these parameters requires highly complicated modeling since incorrect combinations can either lead to overloading of the network, inefficient utilization of the resources, or unacceptable prediction overheads.

1.2 Definition

Among application level transfer tuning parameters, pipelining specifically targets the problem of transferring large numbers of small files. It has two major goals: first, to prevent the data channel idleness and to eliminate the idle time due to control channel conversations in between the consecutive transfers. Secondly, pipelining prevents TCP window size from shrinking to zero due to idle data channel time if it is more than one Round Trip Time (RTT). In this sense, the client can have many outstanding transfer commands without waiting for the “226 Transfer Successful” message. For example, if the pipelining level is set to four in GridFTP, five outstanding commands are issued and the transfers are lined up back-to-back in the same data channel. Whenever a transfer finishes, a new command is issued to keep the pipelining queue full. In the latest version of GridFTP, this value is set to 20 statically by default and does not allow the user to change it. In Globus Online [2], this value is set to 20 for more than 100 files of average 50MB size, 5 for files larger than 250MB and in all other cases it is set to 10. Unfortunately, setting static parameters based on the number of files and file sizes is not affective in most cases, since the optimal pipelining level also depends on the network characteristics such as bandwidth, RTT, and background traffic. Using parallel streams is a very popular method for overcoming the inadequacies of TCP in terms of utilizing the high-bandwidth networks and has proven itself over socket buffer size tuning techniques [3], [4], [5], [6], [7], [8]. With parallel streams, portions of a file are sent through multiple TCP streams and it is possible to achieve multiples of the throughput of a single stream. Setting the optimal parallelism level is a very challenging task and several models have been proposed in the past [9], [10], [11], [12], [13], [14], [15], [16]. The Mathis equation[17] states that the throughput of a TCP stream(BW) depends on the Maximum Segment Size(MSS), Round Trip Time(RTT), a constant(C) and packet loss rate(p).

$$BW = (MSS \times C) / (RTT \times \sqrt{p}) \quad (1)$$

As the packet loss rate increases, the throughput of the stream decreases. The packet loss rate can be random in under-utilised networks however when there is congestion, it increases dramatically. In [9], a parallel stream model based on the Mathis equation is given.

$BW_{agg} <= (MSS \times C) / RTT\ [1 / \sqrt{p1} \ldots 1 / \sqrt{pn}] = n\ (MSS \times C)/(RTT \times \sqrt{p})$ (2)

However excessive use of parallel streams can increase the packet loss rate dramatically, causing the congestion avoidance algorithm of TCP to decrease the sending rate based on the losses encountered. Therefore, the packet loss happening in our case occurs due to congestion. In our previous study[11], we presented a model to find the optimal level of parallelism based on the Mathis throughput equation. Therefore, in Globus Online[2], the parallelism level is set to 2, 8 and 4 respectively for the cases mentioned in the second paragraph above. In the context of transfer optimization, concurrency refers to sending multiple files simultaneously through the network channel. In [18], the effects of concurrency and parallelism were compared for large file transfers. Concurrency is especially good for small file transfers, and overcoming end system bottlenecks such as CPU utilisation, NIC bandwidth, parallel file system characteristics[19](e.g. Lustre file system distributes files evenly and can provide more throughput in multi-file transfers). The Stork data scheduler [20], [21] has the ability to issue concurrent transfer jobs and in most of the cases, concurrency has proven itself over parallelism. Another study [22], adapts the concurrency level based on the changes in the network traffic, does not take into account the other bottlenecks that can occur on the end systems. Globus Online sets this parameter to 2 along with the other settings for pipelining and parallelism. A full comparison of pipelining, parallelism and concurrency is presented in Figure 1, to indicate the differences of each method from each other and from a non-optimized transfer. In this study, we provide a step-by-step solution to the problem of big data transfer bottleneck for scientific cloud applications. First, we provide insight into the working semantics of application-level transfer tuning parameters such as pipelining, parallelism and concurrency. We show how to best utilize these parameters in combination to optimize the transfer of a large dataset. In contrast to other approaches, we present the solution to the optimal settings of these parameters in a question-and-answer fashion by using experiment results from actual and emulation testbeds. As a result, the foundations of dynamic optimization models for these parameters are outlined, and several rules of thumbs are identified. Next, two heuristics algorithms are presented that apply these models. The experiments and validation of the developed models are performed on high-speed networking testbeds and cloud networks. The results are compared to the most successful and highly adopted data transfer tools such as Globus Online and UDT [23]. It has been observed that our algorithms can outperform them in majority of the cases.

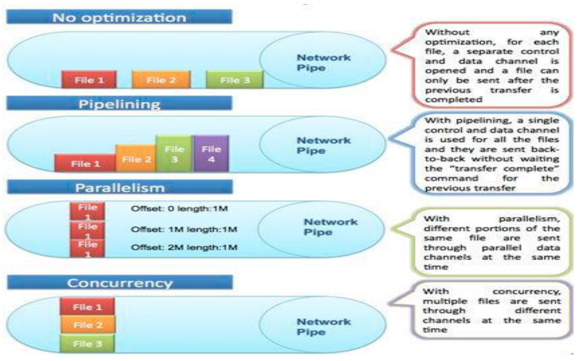

No optimization
File 1
File 2
File 3
Network Pipe
Without any optimization, for each file, a separate control and data channel is opened and a file can only be sent after the previous transfer is completed
Pipelining
File 1
File 2
File 3
File 4
Network Pipe
With pipelining, a single control and data channel is used for all the files and they are sent back-to-back without waiting the "transfer complete" command for the previous transfer
Parallelism
File 1
File 1
File 1
Offset: 0 length:1M
Offset: 1M length:1M
Offset: 2M length:1M
Network Pipe
With parallelism, different portions of the same file are sent through parallel data channels at the same time
Concurrency
File 1
File 2
File 3
Network Pipe
With concurrency, multiple files are sent through different channels at the same time

Fig NO-1 Protocol Comparision

2. LITERATURE REVIEW

The following are the existing system implemented as follows-

- **FTP**

The classic File Transfer Protocol (FTP) defines a capability for clients to send data to servers using the underlying TCP/IP protocol of the public Internet. FTP is an open standard and widely deployed in almost every operating system, is well documented and is broadly accepted as the de facto data movement mechanism available in modern software.

Our studies showed that FTP exhibited low scalability in large volume transfers performed over a Wide Area Network (WAN). It was highly reliable; we did not record a single fault or operational failure in 20 hours of testing, and over 100 gigabytes (GBs) of data movement. FTP's operational and implementation costs are both negligible (e.g., both 0 on our scale).

- **SCP**

The Secure Copy Protocol (SCP) [11] allows for the secure transfer of files between two machines using the SSH protocol for authentication and encryption over a network. Akin to FTP, SCP is built on top of the underlying TCP/IP protocol and is a public, open standard.

SCP, like FTP, exhibited poor scalability over the WAN in our testing results. SCP was highly reliable, with no faults incurred in over 20 hours of testing, using the same FTP datasets. The operational and implementation costs of SCP are both negligible, comparable to that of FTP.

- **GridFTP**

GridFTP [12] extends the FTP protocol with new features required for large volume, fast data transfer, such as striping, partial file access and highly parallel stream-based usage of TCP/IP. GridFTP is the basic data movement mechanism provided by the Globus Toolkit [12], the widely adopted software packages for implementing grid-based software applications.

GridFTP's scalability was extremely high, it was able to transfer at a rate that was directly proportional to that of the dataset volume. GridFTP also exhibited high reliability, with no faults experienced. We found GridFTP to be difficult to deploy, requiring the setup of certificate management for hosts, users, and services. Because of this, we estimate that the implementation costs to be medium (around 3). On the other hand, the operational costs would be negligible (0), as the system is highly reliable and configurable as soon as it is deployed.

- **bbFTP**

bbFTP [13] is an open-source parallel TCP/IP data movement technology. bbFTP's main capabilities are the ability to use SSH and certificate based authentication, on-the-fly data compression and customizable time-outs.

bbFTP is bleeding edge with little industry adoption, and documentation. We found bbFTP to be highly scalable, and also highly reliable (no faults during testing) and configurable. Cost to implement is negligible (0), as bbFTP was easy to find, download, configure and install. Its operational costs are somewhat higher (around 3), due to the small amount of user documentation provided.

- **UFTP**

UFTP [5] or UDP-based file transfer protocol with multicast is an open-source data movement mechanism designed for efficient and reliable transfer of large amounts of data to multiple receivers simultaneously. UFTP is particular effective over a satellite link (with two way communication), or over high-delay Wide Area Networks (WANs) where the reliability mechanisms in TCP/IP severely under-utilize the available network throughput capabilities.

We found UFTP to be highly scalable in the only environment that we were able to test it in (the LAN), outperforming both Aspera and FTP and SCP on similar datasets and volumes. UFTP exhibited extremely poor reliability with the fault rate a function of the total dataset volume. UFTP is a bleeding edge technology, with a small customer base and little documentation. Though it was not difficult to install, it was extremely difficult to configure its firewall rules. Because of this, we estimate high (4) implementation costs, and higher (5) costs to operate it.

- **Aspera**

Aspera is a commercially available product built on top of a proprietary protocol which fully utilizes the capabilities of UDP to send bursts of data from one place to another. Aspera builds on top of the SCP protocol to provide secure, reliable, and most importantly fast transfer of voluminous data sets independent of network latency and packet loss.

We found that Aspera was not scalable, experiencing low transfer rates (nearly linear) based on dataset volume size. The fault rate in Aspera was negligible. Aspera was easy to deploy, and comes as an installer package for most operating systems. Aspera's documentation was fairly poor. Like UFTP, we could not discern the appropriate firewall rules to transfer data using Aspera in the WAN environment. We estimate the cost to implement Aspera to be low, however, thecost to operate it very high.

3. PROBLEM DEFINATION & SCOPE

3.1 Existing System and Its Disadvantages

We review the requirements that motivated our design.
Striping.
Continued commoditization of end system devices means that data sources and sinks are often clusters. Whether data is obtained from disk, sensors, or computation, the "end system" that drives a wide area link may involve many physical devices and considerable internal parallelism. This parallelism may also extend to the external network interface: a common configuration might have individual nodes connected by 1 Gbit/s Ethernet connections to a switch that is itself connected to the external network at 10 Gbit/s or faster. Thus, we wish to support striped data movement operations, in which data distributed across, or generated by, a set of computers or storage systems at one end of a network is transferred to another remote set of storage systems or computers.
Collective operations.
While one can in principle express a data transfer between two clusters as a set of independent point-to-point transfers, it can be valuable to express such transfers as a single "collective" operation. Such an expression can permit a more concise description of the data transfer and provide a convenient logical unit for monitoring and management. Such an expression can also expose opportunities for optimization that might not be apparent in a set of point-to-point transfers. Thus, we wish to treat striped transfers as collective operations.
Uniform interfaces.
Data sources and sinks come in many shapes and sizes, and may include clusters with local disks, clusters with parallel file systems, archival storage systems (with or without parallel data mover support), and geographically distributed data sources.

We want to make it possible for clients to access such sources and sinks via a uniform interface. We also want to make it easy to adapt our system to support different sinks and sources.
Network protocol issues.
The standard protocol for network data transfer remains TCP. However, TCP's congestion avoidance algorithm can lead to poor performance, particularly in default configurations and on paths with high round trip times. Solutions to this problem include careful (ideally automated) tuning of TCP parameters [18], TCP protocol improvements [40], multiple "parallel" TCP connections [28, 46], and the substitution of alternative protocols [13, 14, 27, 33]. We want to support such alternatives. End-to-end performance. Depending on context, high end-to-end performance can require the integrated management of many different devices, including storage systems, computers used to transform data, network interfaces, and network paths, and also perhaps other devices such as computers and storage systems located at intermediate points in a network.
We would like to provide a framework within which a range of such end-to-end management approaches can be applied in a convenient manner.
Diverse failure modes. Collective operations, striped transfers, and end-to-end management offer opportunities for enhanced performance, but also introduce new failure modes. Our design must address robustness and fault tolerance.

3.2 Proposed System

We provide a step-by-step solution to the problem of big data transfer bottleneck for scientific cloud applications. First, we provide insight into the working semantics of application-level transfer tuning parameters such as pipelining, parallelism and concurrency. We show how to best utilize these parameters in combination to optimize the transfer of a large dataset. In contrast to other approaches, we present the solution to the optimal settings of these parameters in a question-and-answer fashion by using experiment results from actual and emulation testbeds. As a result, the foundations of dynamic optimization models for these parameters are outlined, and several rules of thumbs are identified. Next, two heuristics algorithms are presented that apply these models. The experiments and validation of the developed models are performed on high-speed networking testbeds and cloud networks. The results are compared to the most successful and highly adopted data transfer tools such as Globus Online and UDT [23]. It has been observed that our algorithms can outperform them in majority of the cases.

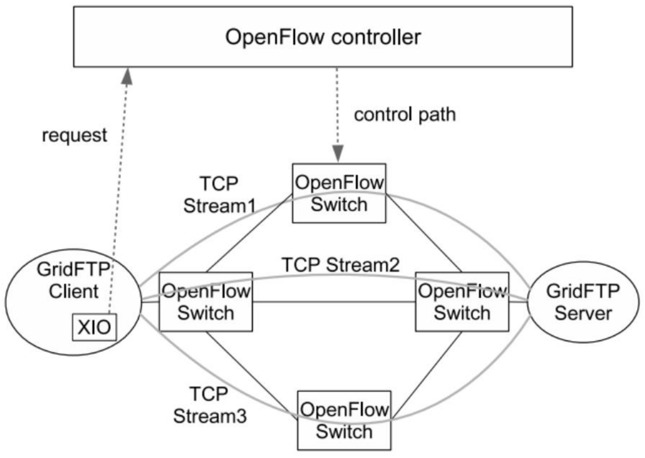


FIG NO-2 Block Diagram of GFTP

The Globus striped GridFTP system aims for (a) modularity, to facilitate the substitution of alternative mechanisms and use in different environments and configurations, and (b) efficiency, in particular the avoidance of data copies. As in systems such as the xKernel [32], we achieve these goals via an architecture that allows a protocol processing pipeline to be constructed by composing independent modules responsible for different functions.

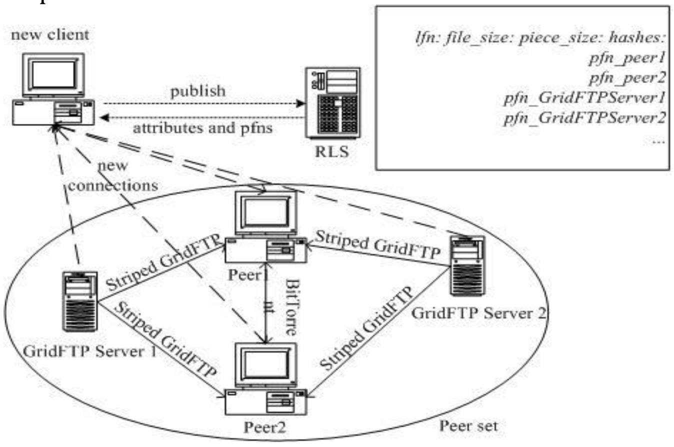


FIG NO-3 Block Diagram of System Architecture

The implementation (Figure 1) comprises three logically distinct components: client and server protocol interpreters (PIs), which handle the control channel protocol (these two functions are distinct because the protocol exchange is asymmetric), and the data transfer process (DTP), which handles the accessing of the actual data and its movement via the data channel protocol. These components can be combined in various ways to create servers with different capabilities. For example, combining the server PI and DTP components in one process creates a conventional FTP server, while a striped server might use one server PI on the head node of a cluster and a The DTP itself is further decomposed into a threemodule pipeline (Figure 2). The data access module provides an interface to data source(s) and/or sink(s). The data processing module performs server-side data processing, if requested by an extended store/retrieve (ESTO/ERET) command. Finally, the data channel protocol module reads from, and/or writes to, the data channel. This basic structure allows for a wide variety of systems, from simple file server logic (data access module reads/writes files, data processing module does nothing, data channel protocol module writers/reads the data channel) to more complex and specialized behaviors (e.g., data module generates data dynamically in response to user requests).

4. METHODOLOGY

This section describes the overlay network topology and methodology used to evaluate the performance of the system.

4.1 GridFTP Overlay Network Topology

The GridFTP overlay network topology used for experiments is shown in Figure 2. It is comprised of ten hosts including three hosts from the University of Calgary Grid Research Centre (GRC), five hosts from Westgrid, one host from ACEnet and one host from the University of Houston which is part of the HP CCN Grid. WestGrid and ACEnet are high performance computing consortia in Western Canada and Atlantic Canada respectively. The HP CCN Grid is an HP led grid computing collaboration that the GRC is also a participant of.
Split servers were placed at hosts grc15 and octarine in the GRC domain as well as condor in the WestGrid domain. Connectivity between domains is provided by CA*net4 in Canada and Abilene in the United states. The majority of hosts are high performance computing facilities.

4.2 GridFTP Log Generation

Initially some of the sites used in these experiments, had no GridFTP traffic between them. For this reason GridFTP logs were created by transferring data at random intervals between the sites. This also allowed us to examine the difference between memory to memory transfers and disk to disk transfers.
The arrival times of the GridFTP transfers were modelled as a Poisson process for each source with an average inter-arrival time of 30 minutes and a randomly selected destination host. These transfers alternated between disk to disk (D2D) transfers and memory to memory (M2M) transfers. This meant that the throughput for transfers between each host pair was measured on average once every 9 hours. For D2D transfers a file size was chosen such that the transfer would take approximately 90 seconds. The range of file sizes available were all powers of 2 from 32MB to 1024 MB. A transfer duration of 90 seconds was chosen because preliminary experiments determined this was sufficient to achieve steady state behaviour.
D2D transfers were intended to include the cost of disk access because not all data transfers are limited by network congestion. In many systems, particularly those that use high bandwidth networks, the bottleneck for the transfer may occur during reading or writing of data to and from disk. The use of caching on disk systems means that what was intended to be a D2D transfer may have in fact been M2D, D2M or M2M. Tests were undertaken and the use of random files on some hosts was implemented in order minimize disk caching effects. Disk caching is only an issue when the disk, not the network is a bottleneck.
All transfers were performed using a GridFTP client built for the purposes of these experiments using the API distributed with the Globus Toolkit. The client reported performance information every time it was received from the destination server. Time measurements for throughput were taken using the performance plugin from the GridFTP libraries. Connection negotiation time was taken into account.
C. Analysis of split point selection
Possible splits for all host pairs were evaluated at random intervals with an exponentially distributed inter-arrival time with a mean of 6 hours. The possible splits were evaluated based on the D2D and M2M throughput predictions. On any connection in which a split connection had higher estimated bandwidth than the direct connection for either the M2M or D2D predictors, a split-connection and a control connection were performed one after the other. The control connection was a single connection from the source to the destination. If the M2M and D2D predictions chose different split points, both were attempted.
Adaptive PCP Algorithm
This algorithm sorts the dataset based on the file size and divides it into 2 sets; the first set (Set1) containing files with sizes less than BDP and the second set (Set2) containing files with sizes greater than BDP. Since setting different pipelining level is effective for file sizes less than BDP (Rule 2), we apply a recursive chunk division algorithm to the first set which is outlined in the following subsection. For the second set we set a static pipelining level of 2.
Recursive Chunk Division for Optimal Pipelining
This algorithm is mean-based to construct clusters of files, with each cluster (chunk) having a different optimal pipelining value. The optimal pipelining level is calculated by dividing BDP to the mean file size and the data set is recursively divided by the mean file size index while several conditions are met. The first condition is that a chunk can only be divided further if its optimal pipelining is not the same as its parent chunk. Secondly, a chunk cannot be less than a preset minimum chunk size and the last rule is that the optimal pipelining level set for a chunk cannot be greater than the preset maximum pipelining level.
The outline is presented in Algorithm 1.

4.3 Algorithm 1 Recursive Chunk Division(RCD)

```
Require: list of files _ start index _ end index _ total number of files _ min chunk size _ parent pp _ max pp
Calculate mean file size
Calculate current opt pp
Calculate mean file size index
if current opt pp! = 1&
current opt pp 6= parent pp &
current opt pp <= max pp &
start index < end index&
mean file size index > start index&
mean file size index < end index&
current chunk size > 2 ⇤ min chunk size then
call RCD dividing the chunk by mean index
(start index− > mean index)
call RCD dividing the chunk by mean index
(mean index + 1− > start index)
else
opt pp = parent pp
end if
```

5. RESULT

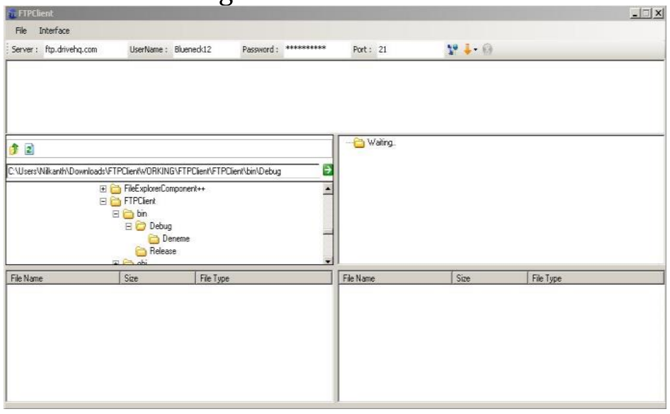


FIG-1 Establishing Connection

Server : ftp.drivehq.com UserName : Blueneck12 Password : *********** Port : 21

C:\Users\Nilkanth\Downloads\FTPClient\WORKING\FTPClient\FTPClient\bin\Debug

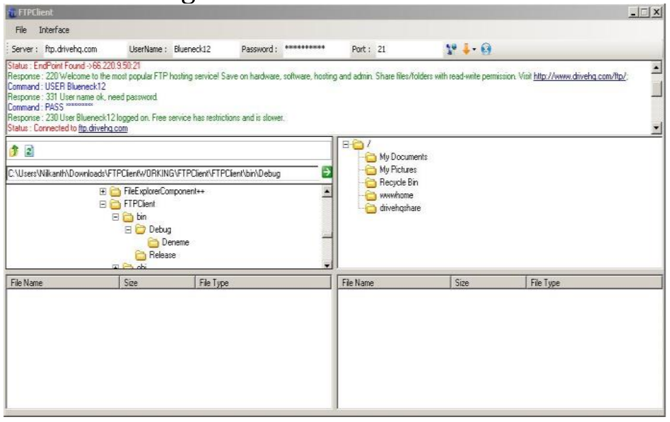


FIG-2 After Login

Server : ftp.drivehq.com UserName : Blueneck12 Password : *********** Port : 21

```
Status : EndPoint Found ->66.220.9.50:21
Response : 220 Welcome to the most popular FTP hosting service! Save on hardware, software, hosting and admin. Share files/folders with read-write permission. Visit http://www.drivehq.com/ftp/;
Command : USER Blueneck12
Response : 331 User name ok, need password.
Command : PASS ********
Response : 230 User Blueneck12 logged on. Free service has restrictions and is slower.
Status : Connected to ftp.drivehq.com
```

C:\Users\Nilkanth\Downloads\FTPClient\WORKING\FTPClient\FTPClient\bin\Debug

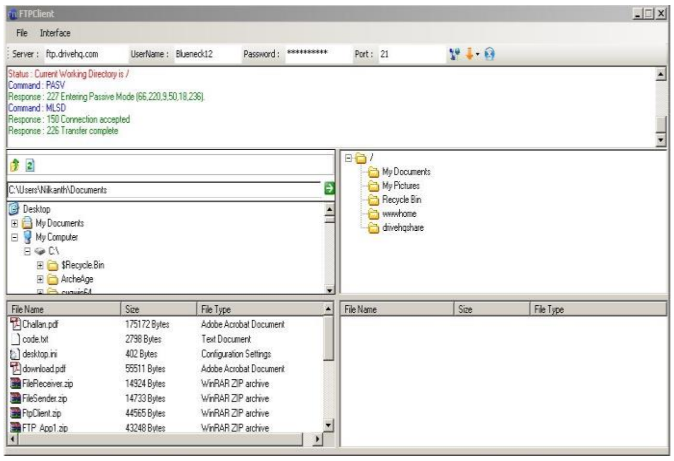


FIG-3 Browse Client

Server : ftp.drivehq.com UserName : Blueneck12 Password : *********** Port : 21

```
Status : Current Working Directory is /
Command : PASV
Response : 227 Entering Passive Mode (66,220,9,50,18,236).
Command : MLSD
Response : 150 Connection accepted
Response : 226 Transfer complete
```

C:\Users\Nilkanth\Documents

File Name	Size	File Type
Challan.pdf	175172 Bytes	Adobe Acrobat Document
code.txt	2798 Bytes	Text Document
desktop.ini	402 Bytes	Configuration Settings
download.pdf	55511 Bytes	Adobe Acrobat Document
FileReceiver.zip	14924 Bytes	WinRAR ZIP archive
FileSender.zip	14733 Bytes	WinRAR ZIP archive
FtpClient.zip	44565 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive

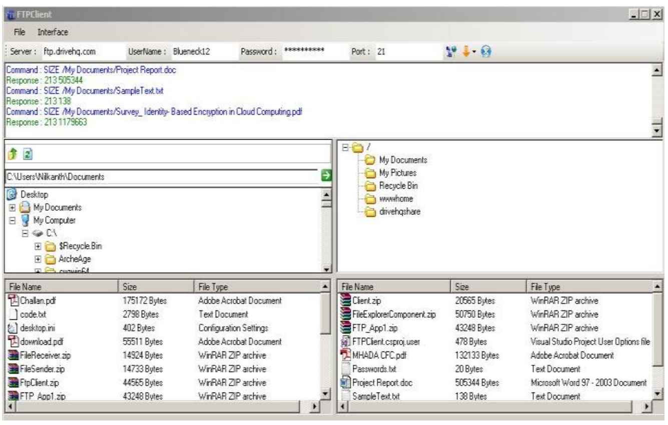


FIG-3 Browse Server

Server : ftp.drivehq.com UserName : Blueneck12 Password : *********** Port : 21

```
Command : SIZE /My Documents/Project Report.doc
Response : 213 505344
Command : SIZE /My Documents/SampleText.txt
Response : 213 138
Command : SIZE /My Documents/Survey_ Identity- Based Encryption in Cloud Computing.pdf
Response : 213 1179663
```

C:\Users\Nilkanth\Documents

File Name	Size	File Type
Challan.pdf	175172 Bytes	Adobe Acrobat Document
code.txt	2798 Bytes	Text Document
desktop.ini	402 Bytes	Configuration Settings
download.pdf	55511 Bytes	Adobe Acrobat Document
FileReceiver.zip	14924 Bytes	WinRAR ZIP archive
FileSender.zip	14733 Bytes	WinRAR ZIP archive
FtpClient.zip	44565 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive

File Name	Size	File Type
Client.zip	20565 Bytes	WinRAR ZIP archive
FileExplorerComponent.zip	50750 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive
FTPClient.csproj.user	478 Bytes	Visual Studio Project User Options file
MHADA CFC.pdf	132133 Bytes	Adobe Acrobat Document
Passwords.txt	20 Bytes	Text Document
Project Report.doc	505344 Bytes	Microsoft Word 97 - 2003 Document
SampleText.txt	138 Bytes	Text Document

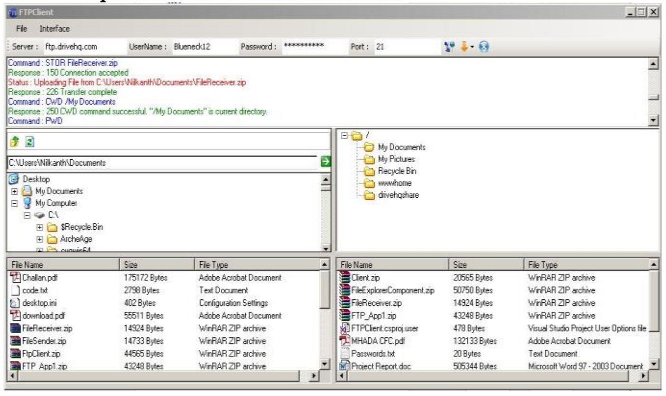


FIG-4 Upload File to Server

Server : ftp.drivehq.com UserName : Blueneck12 Password : *********** Port : 21

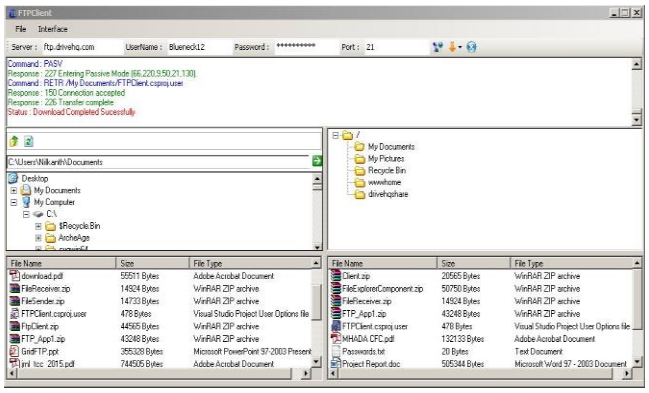
```
Command : STOR FileReceiver.zip
Response : 150 Connection accepted
Status : Uploading File from C:\Users\Nilkanth\Documents\FileReceiver.zip
Response : 226 Transfer complete
Command : CWD /My Documents
Response : 250 CWD command successful. "/My Documents" is current directory.
Command : PWD
```

C:\Users\Nilkanth\Documents

File Name	Size	File Type
Challan.pdf	175172 Bytes	Adobe Acrobat Document
code.txt	2798 Bytes	Text Document
desktop.ini	402 Bytes	Configuration Settings
download.pdf	55511 Bytes	Adobe Acrobat Document
FileReceiver.zip	14924 Bytes	WinRAR ZIP archive
FileSender.zip	14733 Bytes	WinRAR ZIP archive
FtpClient.zip	44565 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive

File Name	Size	File Type
Client.zip	20565 Bytes	WinRAR ZIP archive
FileExplorerComponent.zip	50750 Bytes	WinRAR ZIP archive
FileReceiver.zip	14924 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive
FTPClient.csproj.user	478 Bytes	Visual Studio Project User Options file
MHADA CFC.pdf	132133 Bytes	Adobe Acrobat Document
Passwords.txt	20 Bytes	Text Document
Project Report.doc	505344 Bytes	Microsoft Word 97 - 2003 Document

FIG-5 Download File to Client



Server : ftp.drivehq.com UserName : Blueneck12 Password : ********** Port : 21

```
Command : PASV
Response : 227 Entering Passive Mode (66,220,9,50,21,130).
Command : RETR /My Documents/FTPClient.csproj.user
Response : 150 Connection accepted
Response : 226 Transfer complete
Status : Download Completed Sucessfully
```

C:\Users\Nilkanth\Documents

```
Desktop
  My Documents
  My Computer
    C:\
      $Recycle.Bin
      ArcheAge
```

```
/
  My Documents
  My Pictures
  Recycle Bin
  wwwhome
  drivehqshare
```

File Name	Size	File Type
download.pdf	55511 Bytes	Adobe Acrobat Document
FileReceiver.zip	14924 Bytes	WinRAR ZIP archive
FileSender.zip	14733 Bytes	WinRAR ZIP archive
FTPClient.csproj.user	478 Bytes	Visual Studio Project User Options file
FtpClient.zip	44565 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive
GridFTP.ppt	355328 Bytes	Microsoft PowerPoint 97-2003 Present
jml_tcc_2015.pdf	744505 Bytes	Adobe Acrobat Document

File Name	Size	File Type
Client.zip	20565 Bytes	WinRAR ZIP archive
FileExplorerComponent.zip	50750 Bytes	WinRAR ZIP archive
FileReceiver.zip	14924 Bytes	WinRAR ZIP archive
FTP_App1.zip	43248 Bytes	WinRAR ZIP archive
FTPClient.csproj.user	478 Bytes	Visual Studio Project User Options file
MHADA CFC.pdf	132133 Bytes	Adobe Acrobat Document
Passwords.txt	20 Bytes	Text Document
Project Report.doc	505344 Bytes	Microsoft Word 97 - 2003 Document

This Result shows the retrieved files corresponding to the input files we have selected.

6. APPLICATIONS

The algorithms were tested on real high-speed networking testbeds FutureGrid and Xsede and also cloud networks by using Amazon Web Services 12 EC2 instances. Three different datasets were used in the tests classified as small (between 512KB-8MB), medium (25MB-100MB) and large (512MB-2GB) file sizes. Total dataset size was around 10GB-20GB. The baseline disk write throughput results measured with Bonnie++ and dd for every testbed used in the experiments are presented in Table 2 for comparison purposes.

FutureGrid is a wide area cloud computing testbed with a 10 Gbps network backbone. However it supports only 1 Gbps network interfaces on end-systems so the throughput is bounded by the network interface. Xsede previously known as TeraGrid is also a 10Gbps wide area testbed which is bounded by disk and CPU performance of data transfer nodes. Amazon Web Services (AWS) is a commercial cloud computing services platform that provides a variety of machine instances with different interconnects. On Futuregrid, two clusters Sierra and Hotel which are connected with 76ms RTT were used. On Xsede, Stampede and Gordon were used which are connected with a 50ms RTT. For AWS two compute optimised Linux Ubuntu instances with 10G interconnects were used in the experiments.

6.1 Real Testbed Results

Figure 18.a, b, c shows the results of the algorithms running on FutureGrid testbed. For the small dataset, Globus Online (GO) and UDT perform very poorly. This shows that these tools are not designed for datasets consisting of many small files. PCP algorithm throughput divides the dataset in to 5 chunks and then sets different pipelining levels for each and gradually increases the concurrency level. The final chunk size which is about 8GB results in the highest transfer speed. On the other hand the MC algorithm which is more aggressive in setting concurrency levels outperforms the others in terms of the average throughput.

For the medium dataset, GO keeps up to the MC algorithm throughput by setting static pp, p and cc values. The average throughput of the PCP algorithm follows them and UDT performs the worst. The final chunk of the PCP algorithm which is around 10GB is transferred at the same speed as the MC and GO speeds. The PCP algorithm gradually increases the parallelism level until it no longer increases the throughput. Then it starts increasing the concurrency level with each chunk transfer.

For the large dataset (Figure 18.c), GO and MC average throughput saturates the network bandwidth and UDT performs better. PCPs last two chunks (12GB) reaches the network bandwidth limit. The maximum throughput that can be achieved is bound by the network interface rather than the disk throughput. These results show that our algorithms can reach maximum limit regardless of the dataset characteristics while GO and UDT are only good for relatively large files.

Figure 18.d,e,f presents the experimental results of the algorithms on Xsede network. The same dataset characteristics are used for the tests which are run between SDSC's Gordon and TACC's Stampede clusters.

For the small data set (Figure 18.d) of which file size range is between 512KB and 8MB, MC and PCP algorithms perform the best. The worst results are seen with GO while UDT overperforms it. The last chunk transferred with PCP can adaptively reach 3500Mbps throughput. For the middle dataset the dataset (Figure 18.e) is divided into two. The first set increases the concurrency level while the second set adds parallelism. Again MC algorithm which uses concurrency aggressively performs the best while PCP adaptively learns which concurrency level is best. UDT and GO performs worse. The last chunk transferred with PCP can go beyond 4000Mbps throughput. For the large dataset (Figure 18.f), PCP sets the pipelining level to 2 and applies an adaptive parallelism and concurrency.

The last chunk throughput can reach 4500 Mbps. Again MC and PCP algorithms are the best and can reach maximum disk throughput of Stampede. GO outperforms UDT in this case.

6.2 Cloud Testbed Results

The cloud experiments were conducted using Amazon's EC2 service. Two cpu-optimized c3.8xlarge type nodes with 10G interconnects were launched with an artificial delay of 100ms. Although the interconnects provide 10G bandwidth, the SSD disk volumes bind the maximum achievable throughput to around 1Gbps (Table 2). For the small dataset transfers (Figure 18.g), UDT performs the worst. GO follows UDT with 390Mbps throughput. MC algorithm with a concurrency level of 32 outperforms all others. PCP adaptively reaches 850Mbps throughput with a data chunk

transfer of 7GB but the average throughput of all chunks is around 500Mbps. In the medium dataset (Figure 18.h) GO performs better than PCP average throughput. MC average throughput outperforms all others again. PCP chunk throughput gradually surpasses the others. UDT again performs the worst. For the large dataset (Figure 18.i) GO performance is worse than PCP and MC. It is interesting to see that but since we do not have any control over GO parameters, we do not know why the medium dataset GO results were better. It can be due to different set of pp,p,cc values used for different dataset sizes.
Overall the algorithms that apply our models perform better than GO and UDT in majority of the cases. While PCP algorithm adaptively tries to reach the end-to-end bandwidth, MC algorithms behaves more aggressively based on the initially set concurrency level and both are able to reach maximum achievable throughput.

7. CONCLUSION

Application-level transfer tuning parameters such as pipelining, parallelism and concurrency are very powerful mechanisms for overcoming data transfer bottlenecks
for scientific cloud applications, however their optimal values depend on the environment in which the transfers are conducted (e.g.available bandwidth,
RTT, CPU and disk speed) as well as the transfer characteristics (e.g. number of files and file size distribution). With proper models and algorithms, these parameters can be optimized automatically to gain maximum transfer speed. This study analyzes
in detail the effects of these parameters on throughput of large dataset transfers with heterogenous file sizes and provides several models and guidelines.
The optimization algorithms using these rules and models can provide a gradual increase to the highest throughput on inter-cloud and intra-cloud transfers.
In future work, we intend to write an overhead-free implementation of a GridFTP client to reduce the overhead regarding connection start up /tear down processes for different chunk transfers.

8. RFERENCES

[1] J. S. Hughes and S. K. McMahon, "The Planetary Data System. A Case Study in the Development and Management of Meta-Data for a Scientific Digital Library.," in Proc. of ECDL, pp. 1998.
[2] J. Postel and J. Reynolds, "File Transfer Protocol (FTP) RFC Document, http://www.ietf.org/rfc/rfc959.txt."
[3] W. Allcock, J. Bester, et al., "GridFTP: Protocol Extensions to FTP for the Grid, http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf," RFC Draft Document 2001.
[4] "Aspera Software http://www.asperasoft.com," 2005.
[5] D. Bush, "UFTP - UDP based FTP with multicast. http://www.tcnj.edu/~bush/uftp.html," 2005.
[6] "Designing TeraByte Storage Bricks, http://elib.cs.berkeley.edu/storage/brick/system.html ," 2005.
[7] A. Nayate, M. Dahlin, et al., "Transparent Information Dissemination," in Proc. of Middleware, pp. 2004.
[8] M. Franklin and S. Zdonik, "A Framework for Scalable Dissemination-based systems," in Proc. of OOPSLA, Atlanta, Georgia, pp. 1997.
[9] U. Centintemel and M. Franklin, "Self-adaptive user profiles for large-scale data delivery," in Proc. of ICDE, pp. 622-633, 2000.
[10] "http://www-scf.usc.edu/~mattmann/DM-Matrix-090105.doc," 2005.
[11] "Secure copy - http://en.wikipedia.org/wiki/Secure_copy," 2005.
[12] C. Kesselman, I. Foster, et al., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Intl' Journal of Supercomputing Applications, pp. 1-25, 2001.
[13] "bbFTP - Large files transfer protocol, http://doc.in2p3.fr/bbftp/," 2005.
[14] V. Welch, F. Siebenlist, et al., "Security for Grid Services," in Proc. of Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), pp. 2003.