Spring 2013

Cloud Storage Performance and Security Analysis with Hadoop and GridFTP

Wei-Li Liu
San Jose State University

Cloud Storage Performance and Security Analysis with Hadoop and GridFTP

Wei-Li Liu

Abstract

Even though cloud server has been around for a few years, most of the web hosts today have not converted to cloud yet. If the purpose of the cloud server is distributing and storing files on the internet, FTP servers were much earlier than the cloud. FTP server is sufficient to distribute content on the internet. Therefore, is it worth to shift from FTP server to cloud server? The cloud storage provider declares high durability and availability for their users, and the ability to scale up for more storage space easily could save users tons of money. However, does it provide higher performance and better security features?

Hadoop is a very popular platform for cloud computing. It is free software under Apache License. It is written in Java and supports large data processing in a distributed environment. Characteristics of Hadoop include partitioning of data, computing across thousands of hosts, and executing application computations in parallel. Hadoop Distributed File System allows rapid data transfer up to thousands of terabytes, and is capable of operating even in the case of node failure. GridFTP supports high-speed data transfer for wide-area networks. It is based on the FTP and features multiple data channels for parallel transfers.

This report describes the technology behind HDFS and enhancement to the Hadoop security features with Kerberos. Based on data transfer performance and security features of HDFS and GridFTP server, we can decide if we should replace GridFTP server with HDFS.

According to our experiment result, we conclude that GridFTP server provides better throughput than HDFS, and Kerberos has minimal impact to HDFS performance. We proposed a solution which users authenticate with HDFS first, and get the file from HDFS server to the client using GridFTP.

CONTENTS

GLOSSARY

1-way ANOVA

one-way analysis of variance is a statistical method for comparing means of two or more samples.

CA

it stands for certificate authority. CA issues digital certificates and checks the ownership of a public key by the name subject of the certificates.

FTP

it stands for File Transfer Protocol. It is built on a TCP-based network for file transfer.

generation stamp

Hadoop file system primitive.

GNU

recursive acronym for “GNU’s Not Unix,” an operating system created by Richard M. Stallman in 1984 at MIT’s Artifical Intelligence Lab. This operating system is most commonly used in combination with the Linux kernel, but also used with the Hurd and BSD kernels; with Linux, Hurd, or BSD used as the kernel, the operating system can be referred to as GNU/Linux, GNU/Hurd, or GNU/BSD, respectively.

grid computing

the federation of computing resources from multiple locations to reach a common goal.

GridFTP

extension of the standard File Transport Protocol (FTP) for grid computing environment.

GSI

it stands for Grid Security infrastructure.

Hadoop

a free software framework which can be used for distributed applications.

HDFS

it stands for Hadoop Distributed File System. It is the storage system of Hadoop applications.

KDC

Key Distribution Center is one of the agents in Kerberos authentication. It controls access for the registered users in the realm.

Linux

a kernel which is popularly used with the GNU operating system.

MapReduce

a software framework for processing large data sets in the distributed computing environment.

metadata

it is a form of data which provides detailed information about data.

p-value

it is the probability of getting the test value if the null is true.

POSIX

a name suggested by Richard M. Stallman in response to an IEEE request for a memorable name for IEEE 1003, a family of standards specified by IEEE for maintaining compatibility between operating systems.

SRM

it stands for Storage Resource Management. SRM involves optimizing the efficiency and speed with which a storage area network utilizes available drive space.

X.509

it is a standard for a public key infrastructure.

I. Introduction

Normally, we setup an FTP server when we need to transfer files between hosts. Today, cloud server seems to be a better choice for file transfer, because it provides the same ability with a lot more features. For example, Hadoop not only contains its file system (HDFS), it also provide a platform for people to develop MapReduce programs on top of it. Does FTP server no longer have a place in file sharing? FTP server is easy to set-up using free software, and it allows remote access to the entire hard-drive. Those are the features cloud servers do not have. However, does it provide better performance comparing to cloud servers? Do cloud servers provide enough security to the end users today?

Hadoop is a framework built to support large-scale data-intensive processing. It runs on commodity hardware and used by some of the world's most prominent companies, including IBM, HP, Apple, and Microsoft [1]. In addition to the high-tech companies, Hadoop is also being used for science and academic research.

In section II, we describe the background behind of Hadoop, including HDFS and MapReduce. Moreover, we will introduce Kerberos In section II-D, because we will integrate Kerberos with HDFS to increase its security features. In section II-E, we describe GridFTP by specifying its differences with FTP and some of the GridFTP's features. In section III, we look in details of a research paper and apply some of our work based on their proposed solution design. In section IV, we briefly state the key steps of implementing GridFTP and Kerberos on Hadoop. In section V, we discuss our experiment setup and the performance results of comparing GridFTP with FTP and HDFS.

Once we have all the data available, we come up with a more secured HDFS system with faster file transfer protocol. Based on the current support features, the solution is detailed in section VI. First, client needs to login to the server and calls a bash script. Then, HDFS authenticates the user by matching the user with his password. If the password is correct, the requested file will be copied out of HDFS to local disk and transferred from server to client with GridFTP. After file transfer completes, the local copy is deleted. At the end, a secure file system and faster file transfer can be achieved by the proposed solution.

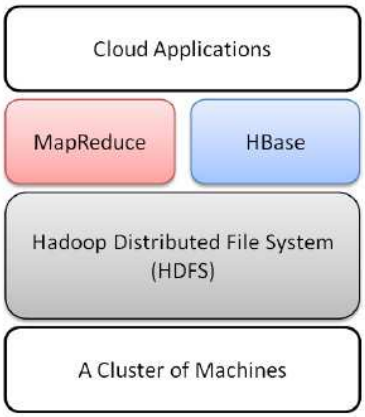

Fig. 1. HDFS is a great object store solution for cloud system

II. Background

A. Introduction of Hadoop

Hadoop is a distributed computing framework which can scale up to thousands of computing nodes and Petabytes of data. If Hadoop users need to scale computation capacity or storage capacity, they just need to add commodity servers to the Hadoop cluster. The preferred operating system is GNU/Linux, with some experimental support for Windows. The name of Hadoop comes from a toy elephant which belongs to the child of the Hadoop's creator, Doug Cutting [1].

Hadoop is made up of two primary components, the Hadoop Distributed File System (HDFS) and the MapReduce engine. HDFS follows master/slave architecture. A HDFS cluster usually contains a single NameNode and a bunch Data Nodes. NameNode is responsible monitoring and distributing data to DataNodes. DataNodes within the same cluster would communicate over the network to balance data blocks, and ensure data is replicated throughout the cluster.

Hadoop's MapReduce was originally based on Google's MapReduce. This type of paradigm processes application by breaking input into small parts, and these parts can be run on any node in a cluster. The MapReduce engine is made up of two main components, Job Tracker and Task Tracker. Users submit jobs to a JobTracker which distributes the task to Task Trackers for data processing.

HDFS has many features which fits data-intensive computing, such as high scalability, reliability and throughput. As Hadoop is increasingly being used on the grid and cloud environment, more companies will invest their research and development in Hadoop.

B. Hadoop Distributed File System (HDFS) Architecture

1) ***HDFS Design Features*** *[2]:*

a. **Hardware failure**

HDFS cluster consists of thousands of server machines, and each of them stores part of the

file system's data. Hardware system failure is very common among HDFS clusters, so failure detection and automatic recovery of nodes are important features of HDFS.

b. **Streaming Data Access**

HDFS emphasizes on high throughput of data access instead of low latency. It is designed for batch processing and requires streaming access to its data sets.

c. **Large Data Sets**

HDFS cluster provides high bandwidth and scalability. It normally runs applications with terabytes file in size.

d. **Simple Coherency Model**

HDFS applications requires write-once-read-many access model (A file cannot be changed once created and written by any particular user) to keep data coherency and enable high data throughput.

e. **Computation nears Data**

It is inefficient to move the data to where the application is running. When the dataset is huge, computation is more efficient if it is processed near the data. It reduces network congestion and increases the throughput.

*2) **NameNode**:* HDFS stores information about files and directories at NameNode , which records attributes like permission, modification and access times. NameNode acts like central coordinator which split file content into small data blocks, and placed them at different DataNodes.

When HDFS client reads a file, it contacts with NameNode first to get the locations of data blocks. Then it reads block content from the DataNode closest to the client. When HDSF client writes data to HDFS DataNode, the client informs the NameNode to select DataNodes to store block replicas, and it writes the data simultaneously. [3].

Besides Namenode, there is SecondaryNameNode for periodic checkpoints. The SecondaryNamenode periodically records current NameNode image, and joins them into a new image to the NameNode. Therefore, if the NameNode fails, you can restart it on the same node without shutting down the entire cluster.

3) ***DataNodes****:* Each block replica consists of two files at the host's native file system. They are the original data and the data block's metadata. Metadata contains information about data block's checksums and generation stamp.

At startup of Hadoop cluster, each DataNode connects to the NameNode and performs a hand-shake. Handshake checks the NameNode's namespace ID and the Hadoop version. The Hadoop cluster will automatically shut down if either does not match, because nodes with a different namespace ID shouldn't join the cluster, and inconsistent version of Hadoop could cause data loss.

DataNodes register with the NameNode with unique storage IDs after the handshake. The storage ID is used to identify the DataNodes, and it remains the same after the registration. In addition, a DataNode sends block report to the NameNode hourly. A block report consists of the block id and the generation stamp. It is a way for DataNode to inform NameNode that block replicas are in its possession.

Moreover, DataNodes send heartbeats to the NameNode every three seconds to confirm that the DataNodes are operating. If the NameNode doesn't receive heartbeats from a DataNode for more than 10 minutes, the NameNode considers that the DataNode is down and creates new data replicas on other DataNodes.

Heartbeats from a DataNode contain information such as storage capacity and the data transfer in progress. NameNode controls load balancing and storage allocation for the DataNodes according to the heartbeats' information. The NameNode sends instructions to the DataNodes by replying to the heartbeats rather than contacting DataNodes directly. Therefore, frequent heartbeats on the HDFS cluster are important in terms of maintaining a well-functioned file system. [3].

4) ***File Read and Write****:* Once a new file is created at HDFS, the file written cannot be changed after the file is closed. The client obtains a lease for the file if the file is opened by the HDFS client. Thus, no other clients can write to the file. The lease is completely revoked when the file is closed.

Failure of nodes is common in a cluster of thousands of nodes. HDFS generates checksums for each data block, and it can be used to detect any file corruption. When HDFS client reads

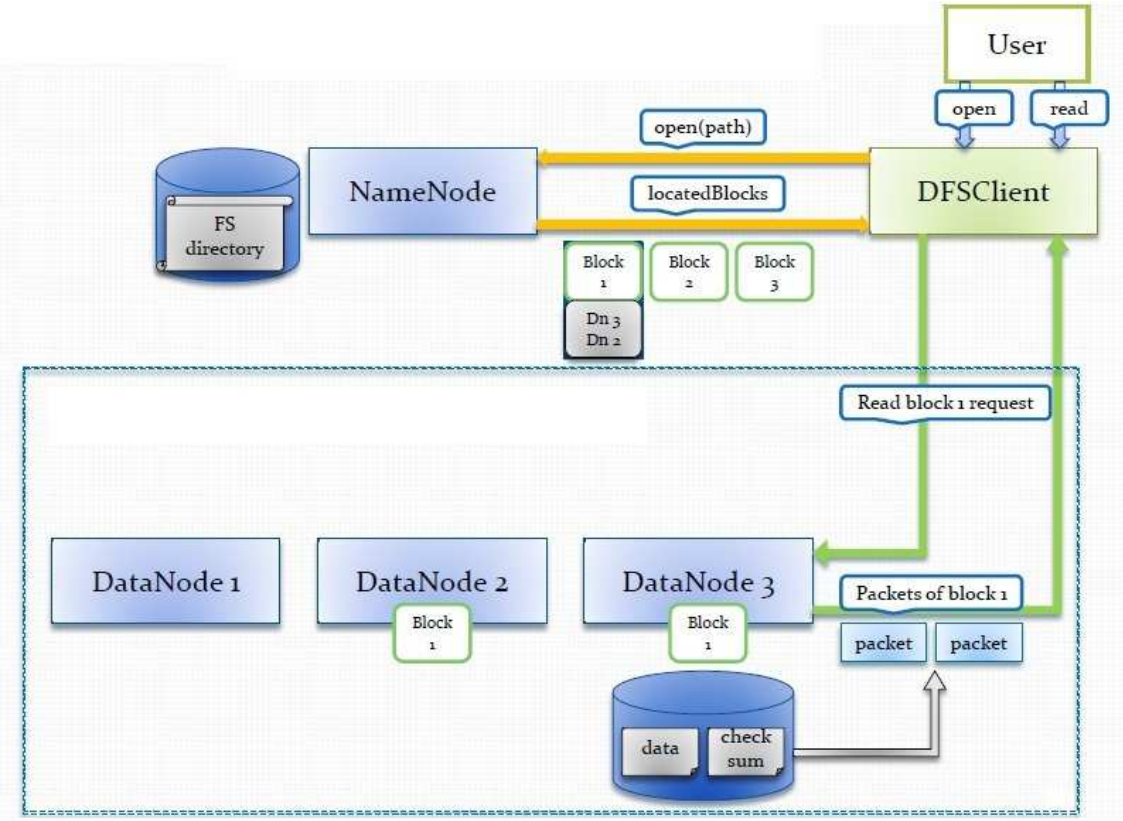

Fig. 2. Read File Process of HDFS

a file, each block's checksums is computed. If the checksum is incorrect, NameNode will get notifications and select a different DataNode to retrieve data blocks [3].

C. MapReduce [2]

1) ***Overview****:* Google adopts MapReduce framework in 2004, and Hadoop is developed based on Google's MapReduce. MapReduce is a paradigm which splits the input data into small pieces and processes by map tasks. Then, it sorts output of the maps and uses it as input to the reduce tasks. Eventually, the master node collects the results of reduce tasks and combines them in a standard form of output.

2) ***Inputs and Outputs****:* MapReduce computational applies data stored either in HDFS or Hbase. The entire process can be summarized as follows:

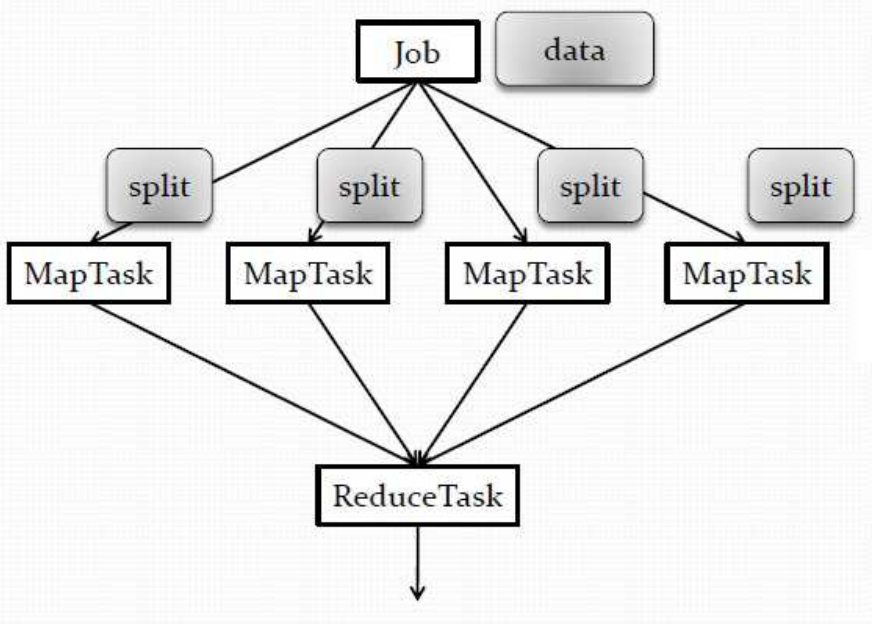

Fig. 3. Logic of MapReduce

MapReduce Steps

1) Copy file into Master and each Worker Machine.

2) Master decides which Worker does Map, and which does Reduce.

3) Put blocks of data into Map computing machine.

4) Store the mapping result to local disk.

5) Read the Mapping result, and execute reduce.

6) Output the final computing result.

D. HDFS security

1) ***Overview****:* HDFS security was based on the POSIX model of users and groups. Each HDFS file and directory contains three different permissions: read, write and execute. Security permissions can be changed by operations such as -chmod or-chown, which are similar to POSIX/Linux commands, but not all of the POSIX commands are available on HDFS. [4]

It is easy to break file permission in HDFS and MapReduce jobs at early Hadoop versions (0.17-0.20). If Hadoop cluster runs behind a firewall, it may not expose to the external attack , but it is still vulnerable to the internal security attack. [5]

To summarize, Hadoop security flaws include

- No authentication for users.
- No authenticate for services.
- No secure network transports.

It presents tons of security issues [6]

- Malicious users could submit jobs and execute the Task Tracker.
- Unknown users can communicate directly with a Datanode and read the block location details.
- Unauthorized users can act as authorized users and access the cluster.
- Exposed to eavesdropping/sniffing on the network.

2) ***Kerberos****:* HDFS (post-1.0) improves on security by adding support of authentication protocol (Kerberos). Kerberos was developed by Massachusetts Institute of Technology (MIT). Kerberos uses central authentication server to control the access and service request of users. The user's password does not have to go through the network, so it avoids possibility of password interception. [7]

Hadoop with Kerberos would authenticate users at master nodes (JobTracker and NameNode), and users use their Kerberos key to communicate securely with the NameNode. After the initial authentication, HDFS uses the delegation tokens to decrease the communication load on Kerberos KDC (Key Distribution Center). [8]

The mechanism for authentication through Kerberos is shown in Figure 4. First, the client authenticates himself to the Authentication Server (AS), which sends the username to a Key Distribution Center (KDC). Then, KDC generates a Ticket Granting Ticket (TGT), which is encrypted by a user's password. If a user enters the correct password, TGT is decrypted and

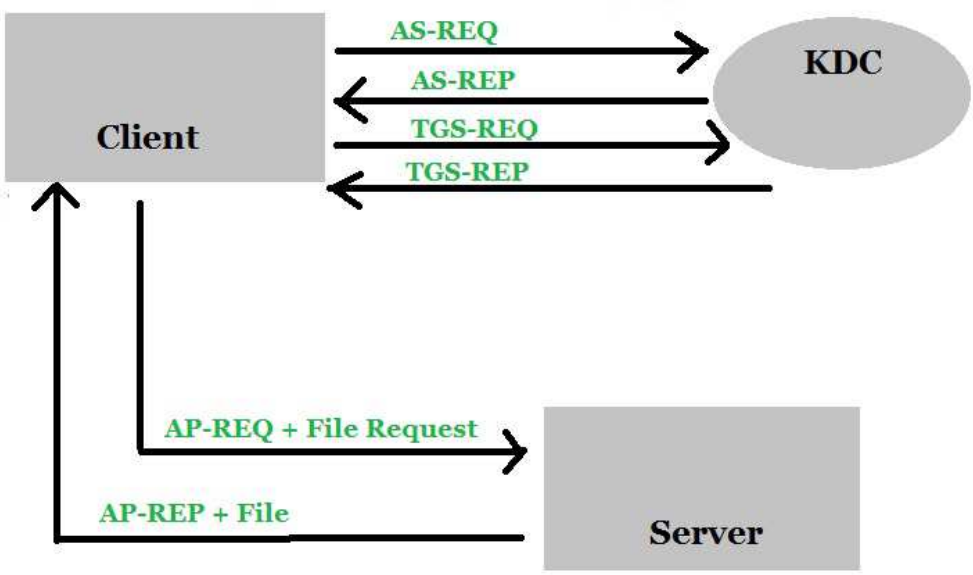

Fig. 4. Kerberos Authentication Steps

user obtains access. A user can use this TGT to request services from Ticket Granting Service (TGS). If TGT is valid, TGS issues the ticket and sessions keys, and the user acquires access to the service. [9]

The following sums up security enhancement of Hadoop with Kerberos

- Hadoop user authentication is secured by Kerberos.
- Hadoop servers can trust the user's identity

E. GridFTP

1) ***Overview****:* GridFTP could perform data transfer between incompatible storage and access systems, because it is an extension of FTP standard. GridFTP provides the following features for secure and fast transfer:

- High performance data transfer by parallel streams
- secure data transfer (GSI) on grid computing applications

The following sections look at the details and features of GridFTP implementations.

2) ***Two-channel Protocol****:* Similar to FTP, GridFTP utilizes two separate channels for communication (a control channel and data channel) (see Fig. 5). The control channel is encrypted and low bandwidth TCP communication. The data channel is high bandwidth communication links of actual data.

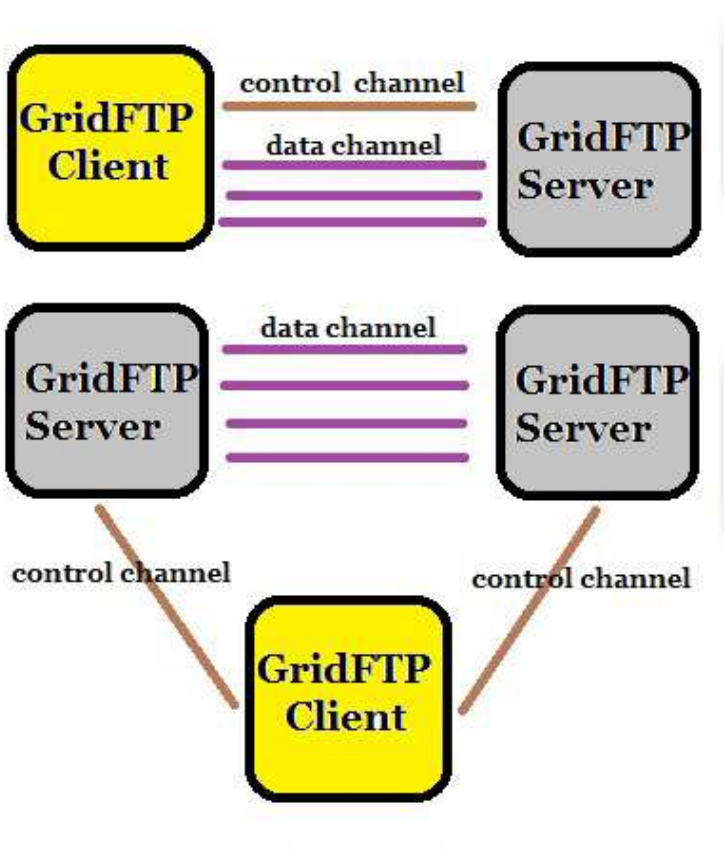

Fig. 5. Two-Channel GridFTP Communication and Third-party Control of Data Transfer

3) ***Third-party control of data transfer****:* GridFTP provides third-party data transfers between servers. It lets user or client to initiate and control of data transfer between two hosts. Fig. 5 shows that the client establishes the control channel with two servers, and the data channel is set up between the servers for actual data transfer. Client is notified when the transfer is done.

4) ***Grid Security Infrastructure****:* GSI uses public key cryptography as the basis for its authentication. The key concept of GSI is the certificate, which is encoded in the X.509 certificate format. CA's main responsibility is to verify the identity of the subject on the certificate, so we must trust the CA. A GSI certificate includes information:

- The person or subject which certificate represents.
- The subject's public key
- The identity of a Certificate Authority (CA).
- The digital signature of the CA

5) ***Parallel data transfer****:* Multiple TCP streams in parallel is so much faster than a single TCP stream. It is main the reason why GridFTP is faster than FTP

6) ***Partial file transfer****:* GridFTP supports commands to transfer arbitrary subsets of file rather than complete file. This feature allows failed large file transfers to transfer the remaining portions of the file instead of starting from beginning.

7) ***Automatic negotiation of TCP buffer/window sizes****:* Optimize TCP window sizes improves data transfer performance. GridFTP supports both manual configurations and automatic negotiation of TCP buffer sizes for file transfer [10].

III. Related Studies

A. *Hadoop Distributed File System for the Grid [11]*

The ability to automatically replicate data on a grid is important for computing sites like those at the Large Hadron Collider (LHC) because the large volumes of high-energy physics experimental data must be shared with collaborators around the globe. The Hadoop Distributed File System (HDFS) provides a reliable method of file replication and distribution that is flexible in terms of architecture of choice at different sites, and has a low cost of deployment and maintenance.

Integrating HDFS seamlessly with the grid requires three components:

- **FUSE module for the Linux kernel**
  FUSE provides a POSIX-like interface to HDFS, which mounts the whole file system ”locally” so that user applications can access data as if it were local storage.
- **GridFTP server**
  GridFTP provides WAN transfer. There is a problem posed by the fact that HDFS supports only synchronous write, so a special GridFTP plugin is required to allow it to assemble asynchronously transferred packets to be sequentially written to HDFS.
- **BeStMan server**
  BeStMan provides an SRMv2 interface to HDFS. SRMv2 is an interface specification for grid-aware Storage Resource Managers (SRMs).

Using the FUSE module solves the issue of direct data access for applications that require a POSIX-compliant file system. Data access via a FUSE mount between two storage elements is facilitated by SRM and GridFTP. The export of HDFS to a machine is subject to considerations of security for the site. Direct access to data is also possible through the Hadoop client. However, it can only be used for file transfer.

Those components are extremely helpful on conducting our experiment. We will be using GridFTP server to transfer data directly from the server to client instead of operating it through SRM, because our experiment only covers nodes in a local area network. Moreover, direct

access to the HDFS data through Hadoop client is needed in our experiment. It gives us clean performance results of HDFS server, so we can compare it with GridFTP server.

In the Large Hardon Collider project, security issues are addressed based on X.509 authentication and authorization. It doesn't solve the underlying security issues of HDFS. The Hadoop framework performed insufficient authentication and authorization of both users and services. Based on the security approach published in 2010 Black Hat Conference, We would use the Simple Authentication and Security Layer with Kerberos to authenticate Hadoop users [12].

IV. Implementation

A. GridFTP

We use Globus-5.0.5 toolkit to install GridFTP in our experiment. GridFTP uses the SSL protocol along with X.509 credentials. We need to configure GSI correctly and build CA. The key steps to setup GSI are summarized as following

(1) Setup GLOBUS_LOCATION path and execute the shell script

```
# export GLOBUS_LOCATION=/sandbox/globus/globus-5.0.5
# source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

(2) Execute perl script to build CA

```
perl gt-server-ca.pl -y
```

Provide CA key(a password at least 4 characters) in the file(gt-server-ca.pl) and CA will generate certificate(hostcert.pem) and associated key(hostkey.pem). Make a /etc/grid-security folder and move the certificate and key to the folder.

(3) The myproxy-server.config file sets the policy for the myproxy-server, specifying what credentials may be stored in the server's repository, who is authorized to retrieve credentials, and other configurable server behaviors.

The following policy enables all credential repository features

```
accepted_credentials "*"
authorized_retrievers "*"
default_retrievers "*"
authorized_renewers "*"
default_renewers "none"
authorized_key_retrievers "*"
default_key_retrievers "none"
trusted_retrievers "*"
default_trusted_retrievers "none"
cert_dir /etc/grid-security/certificates
```

(4) Edit /etc/xinetd.d/myproxy by setting correct GLOBUS_LOCATION path. It is the path where GridFTP is installed.

```
service myproxy-server
{
socket_type = stream
protocol = tcp
wait = no
user = root
server = /sandbox/globus/globus-5.0.5/sbin/myproxy-server
env = GLOBUS_LOCATION=/sandbox/globus/globus-5.0.5
LD_LIBRARY_PATH=/sandbox/globus/globus-5.0.5/lib
disable = no
}
```

(5) Add admin username (yang) to the certificate

```
myproxy-admin-adduser -c "yang" -l yang
```

It requres user to enter password for the certificate (at least 6 characters). To sign the certificate, you need to enter the password of the CA key (the one you set at step 2). If the password is correct, the credential will be stored.

(6) Edit /etc/xinetd.d/gridftp by setting correct GLOBUS_LOCATION path

```
service gsiftp
{
...
env += GLOBUS_LOCATION=/sandbox/globus/globus-5.0.5
env += LD_LIBRARY_PATH=/sandbox/globus/globus-5.0.5/lib
server = /sandbox/globus/globus-5.0.5/sbin/globus-gridftp-server
...
}
```

(7) Edit /etc/services. Adding ”gsiftp 2811/tcp” under Local services. GridFTP server uses port 2811 for tcp connection.

```
# Local services
myproxy-server 7512/tcp # Myproxy server
gsiftp 2811/tcp
```

(8) Reload the service and start gsiftp

```
/etc/init.d/xinetd reload
$ export GLOBUS_LOCATION=/sandbox/globus/globus-5.0.5
$ source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

(9) Login gridftp server by entering the certificate’s password. If should return ”a credential has been received”

```
$ myproxy-logon -s globusserver
```

(10) Transfer file to local disk using globus-url-copy command line tool

```
$ globus-url-copy gsiftp://globusserver/home/yang/testinput
file:///tmp/testinput
```

B. Hadoop with Kerberos

Two steps to build secure HDFS: build KDC and configure Hadoop with Kerberos authentication. We use Kerberos V5 1.8.3 package for implementation.

(1) Create Kerberos database. Here we name our realm ”HADOOP.LOCALDOMAIN”

```
/usr/local/sbin/kdb5_util create -r HADOOP.LOCALDOMAIN -s
```

(2) Generate HDFS keytab as authentication password. Here we generate ”yang.keytab” and assign ”yang/admin” as the user of this keytab

```
$ sudo kadmin.local
```

```
kadmin.local: ktadd -k /usr/local/var/yang.keytab yang/admin
```

(3) For all the nodes of Hadoop cluster, create file krb5.conf under /etc directory.

```
[libdefaults]
default_realm = HADOOP.LOCALDOMAIN
...
...
dns_lookup_kdc = true
dns_lookup_realm = true
[realms]
HADOOP.LOCALDOMAIN =
{
kdc = kdc
admin_server = kdc
default_domain = localdomain
}
[domain_realm]
.localdomain = HADOOP.LOCALDOMAIN
localdomain = HADOOP.LOCALDOMAIN
```

(4) Edit core-site.xml to enable Kerberos authentication

```
⟨configuration⟩
...
...
⟨property⟩
⟨name⟩hadoop.security.authorization⟨/name⟩
⟨value⟩true⟨/value⟩
⟨/property⟩

⟨property⟩
⟨name⟩hadoop.security.authentication⟨/name⟩
⟨value⟩kerberos⟨/value⟩
⟨/property⟩
⟨/configuration⟩
```

(5) Edit hdfs-site.xml. Configure delegation tokens, provide the name of Kerberos principal and the path of keytab to both namenode and datanode

```
⟨configuration⟩
...
...

⟨property⟩
⟨namedfs.block.access.token.enable⟨/name⟩
⟨value⟩ true⟨/value⟩
⟨/property⟩
...
⟨property⟩
⟨name⟩dfs.namenode.keytab.file⟨/name⟩
⟨value⟩/home/yang/Desktop/hadoop-1.0.2/conf/yang.keytab⟨/value⟩
⟨/property⟩

⟨property⟩
⟨name⟩dfs.namenode.kerberos.principal⟨/name⟩
⟨value⟩yang/_HOST@HADOOP.LOCALDOMAIN⟨/value⟩
⟨/property⟩

⟨property⟩
⟨name⟩dfs.namenode.kerberos.https.principal⟨/name⟩
⟨value⟩yang/_HOST@HADOOP.LOCALDOMAIN⟨/value⟩
⟨/property⟩
⟨/configuration⟩
```

(6) Start HDFS. Secure Hadoop requires root account to start datanode.

```
bin/hadoop-daemon.sh start datanode
```

(7) Contact KDC and enter the password correctly to get the key. Only registered user of KDC could enter HDFS.

```
kinit
```

(8) Here is Java code snippet for accessing HDFS from client

```
public void readFile(String file) throws IOException
{
      Configuration conf = new Configuration();

      conf.addResource(new Path("/home/yang/Desktop/core-site.xml"));
      conf.addResource(new Path("/home/yang/Desktop/hdfs-site.xml"));

      UserGroupInformation.
          loginUserFromKeytab("yang/globusclient@HADOOP.LOCALDOMAIN",
             "/home/yang/Desktop/yang.keytab");
      //Assume Kerberos is enabled, we login using
         dfs.namenode.keytab.file
      //yang/globusclient@HADOOP.LOCALDOMAIN has the access right to the
         HDFS

      UserGroupInformation.setConfiguration(conf);

      String uri = "hdfs://test:9000/"; //HDFS URL

      FileSystem fileSystem = FileSystem.get(URI.create(uri),conf);

      Path path = new Path(file);

      FSDataInputStream in = fileSystem.open(path);
      ...
      ...
}
```

V. Experiment

A. Environment Setup

The goal of the experiment is comparing the performance of GridFTP server with the cloud-based HDFS server. We use Oracle Virtualbox to get our nodes and cluster setup. Every node in the cluster uses the same specs (memory, OS, etc.) except HDFS slaves and KDC. Slaves of the HDFS cluster are only responsible for storing data, and handles request from master, so the 512 MB of memory should be sufficient. KDC is the Key Distribution Center. The primary duty is authenticating HDFS users and services. It doesn't require high loads of computation either. The following are the specs of the host platform and the guest platform.

Host Machine

- Model: ThinkPad T420s
- Processor: Intel Corei5-2540M, CPU@2.6GHz
- RAM: 8.00GB
- System type: 64bit OS
- Operating System: Windows 7

Virtual Machine

(a) GridFTP Server, GridFTP Client, HDFS cluster (Master), HDFS Client, Secure HDFS, Single-node HDFS

- Software: Oracle Virtualbox 4.1.18 VMs
- Base memory: 2048MB
- System type: 64bit OS
- Operating System: Linux Ubuntu 10.04
- Network Adapter: Intel PRO/1000 MT Desktop

(b) HDFS cluster (Slaves), KDC

- Software: Oracle Virtualbox 4.1.18 VMs

- Base memory: 512MB
- System type: 64bit OS
- Operating System: Linux Ubuntu 10.04
- Network Adapter: Intel PRO/1000 MT Desktop

We set up a separate and clean virtual network to connect all the VMs together. There are total 3 different comparisons we will be testing in our experiments. First of all, we want to prove if GridFTP is indeed faster than FTP. Then, we will use GridFTP server to be our choice of FTP server, and HDFS server as the cloud storage to compare the performance of these two. Based on the results, we will combine these two to get the best of each. Finally, after the integration of Kerberos with Hadoop, we would like to check if the security enabled HDFS suffers in performance. We will run the experiment with 3 different sizes of file (1GB, 3GB, and 5GB) with null character.

B. FTP v.s. GridFTP

	FTP	**GridFTP**
Security	Inherently insecure. Rely on SSH and SSL for security	Provides authentication and encryption to file transfer
Performance	File transfer using single TCP stream	Multiple simultaneous TCP stream from the source to make better use of bandwidth
Third party transfer	Allows remote transfer between servers to be initiated by local client	Adds security and authentication for the local initiator
Partial file transfer	It doesn't support the transmission of a certain portion of a file	Allows a subset of a file to be sent

1) ***Design****:* We set up the FTP server and GridFTP server on one VM, and FTP client and GridFTP client on another VM. We use Globus toolkit for installing GridFTP server, and vsftpd(very secure FTP Daemon) package for FTP server. The Globus project is actively developing

tools for applications in a grid environment, and vsftpd is a GPL license FTP server for Unix system. We transfer the same size of the file 5 times and compute the average time for each size.

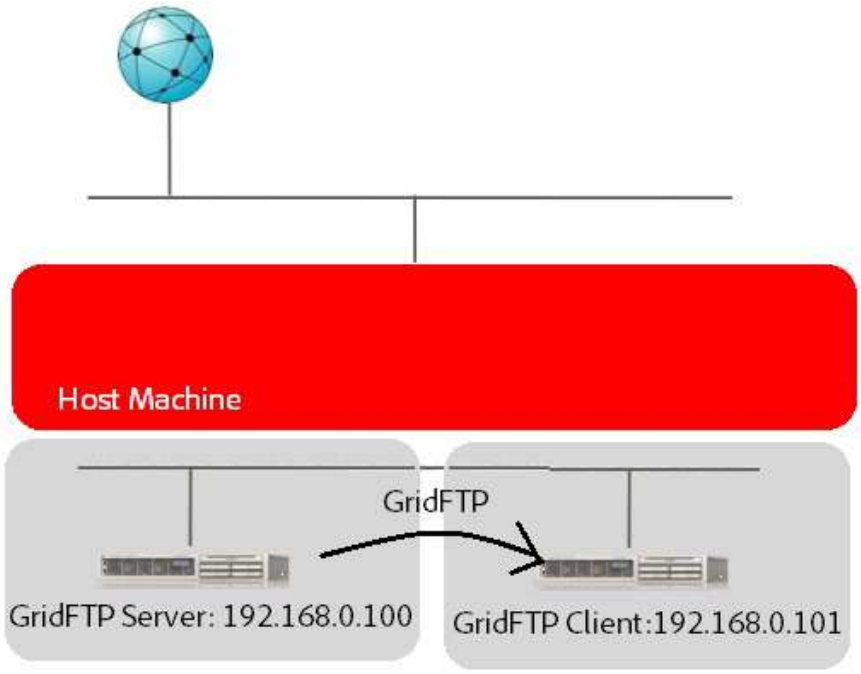

Fig. 6. GridFTP Experiment Diagram

File Size	1GB	3GB	5GB
1	18.59	48.97	97.01
2	19.34	51.34	94.23
3	17.53	47.68	93.88
4	17.44	46.52	95.45
5	18.11	47.36	95.21
Avg.	**18.22**	**48.07**	**95.16**

TABLE I
FTP PERFORMANCE (IN SECONDS)

File Size	1GB	3GB	5GB
1	8.71	27.58	53.19
2	8.70	25.27	51.78
3	9.1	24.87	50.31
4	8.35	25.19	54.81
5	8.49	26.87	53.32
Avg.	**8.67**	**25.96**	**52.68**

TABLE II
GRIDFTP PERFORMANCE (IN SECONDS)

2) ***Analysis****:* The results give us strong evidence that GridFTP is faster than FTP. The average speed for 1G, 3G, 5G are significant faster for GridFTP. It has to do with parallel TCP streams and automatic TCP optimization as we stated earlier. GridFTP enhances its security feature with GSI, but the performance doesn't suffer.

The network traffic captured from ”WireShark” could explain why GridFTP is faster than FTP. GridFTP uses 2 TCP streams. For the first stream, GridFTP server uses a port 2811 for control channel. It will connect with a GridFTP client with a dynamic port number. The second stream is the data channel. Both GridFTP server and GridFTP client uses a dynamic port for data transfer. GridFTP balances the traffic load by using different port number for control and data channel. As a result, it achieves better transfer speed. On the other hand, FTP server and FTP client use a single port to handle both control and data channel (port 20 for FTP server and a dynamic port for FTP client), so it is not as fast as GridFTP.

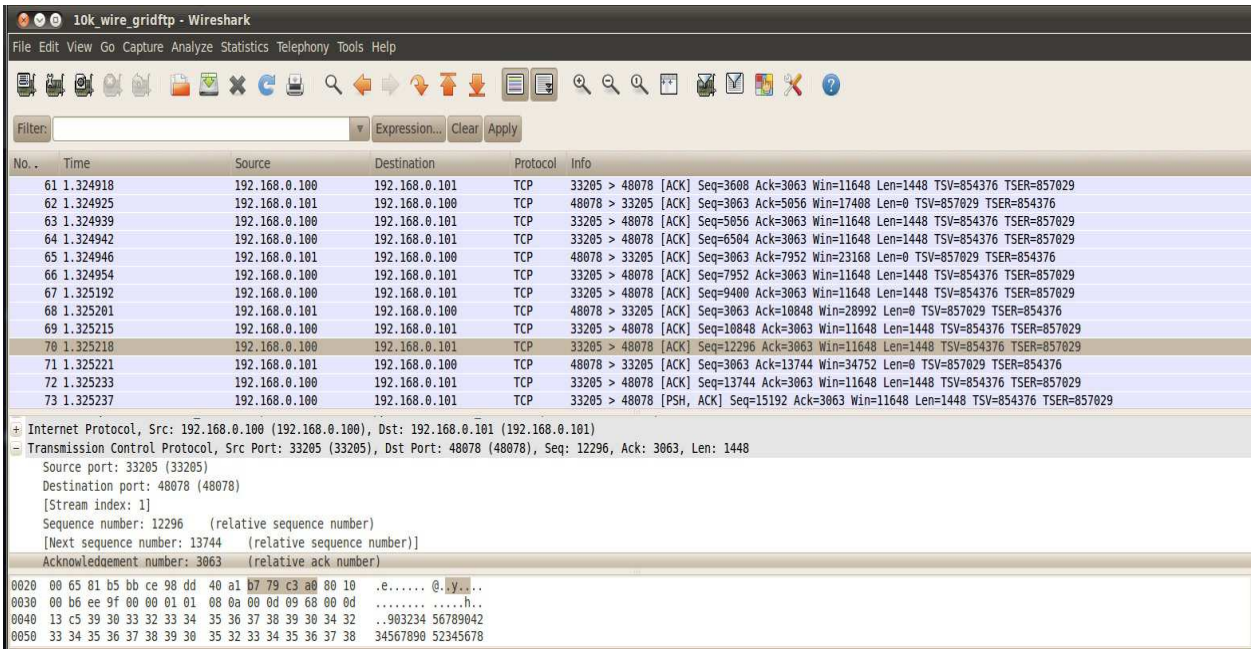

Fig. 7. GridFTP WireShark Captured Result Shows that Not Every Data Packet Requires an ACK from Client

In addition to parallel streams, we also notice that for each ack packet from the GridFTP client, GridFTP server can send multiple data packets. In contrast, FTP client has to send ack for every data packet. Less number of packets also contributes to much better performance of GridFTP. The advertised window of GridFTP doesn’t seem to have much effect on performance. Although the client declares window size as large as 50000 bytes to server, server can only transfer 1448 bytes of data due to the Ethernet default packet size limit (1500 bytes).

Traditional FTP server is easy and cheap to setup, and it provides a convenient method for data transfer. However, if the goal is to have faster and reliable transfer, GridFTP should be

the choice. With that conclusion, we move on to our major research topic, comparing GridFTP server with HDFS server.

C. HDFS Cluster v.s. GridFTP Server

	HDFS Cluster	**GridFTP Server**
Security	No Authentication on services and users. No encryption on data transport	Provides authentication and encryption to file transfer
Performance	Increases throughput by utilizing computing power from multiple nodes	Multiple simultaneous TCP stream from the source to make better use of bandwidth
Data Access	Users can only access data from HDFS	Users can access the entire disk
Fault Tolerant	Data block is replicated 3 times for fault tolerance	Transfer can be automatically restarted if a server problem occurs
Cost	Cheap to set up as multiple nodes can be built on a single host	Hardware cost could be significant for a large scale experiment
Other Features	Capable of more operations such as running MapReduce programs	Simple data storage system for grid computing

1) ***Design****:* We use Apache Hadoop 1.0.2 to setup our HDFS Cluster. The cluster contains 3 nodes in total, one act as master and the other two are slaves. The master node will be the NameNode as well as the DataNode in the experiment. Both slaves would serve solely as DataNodes for storing data blocks. Based on this design, we have 3 replicas for every data block. On the client side, a small Java HDFS client program with Eclipse Hadoop plugin is written for interacting with HDFS. As soon as everything is ready, the client could initiate file transfer with HDFS through JAVA RPC (Remote Procedural Call). The architecture of the HDFS cluster is demonstrated in Fig. 8. Time of transfer is based on execution time of the HDFS client program.

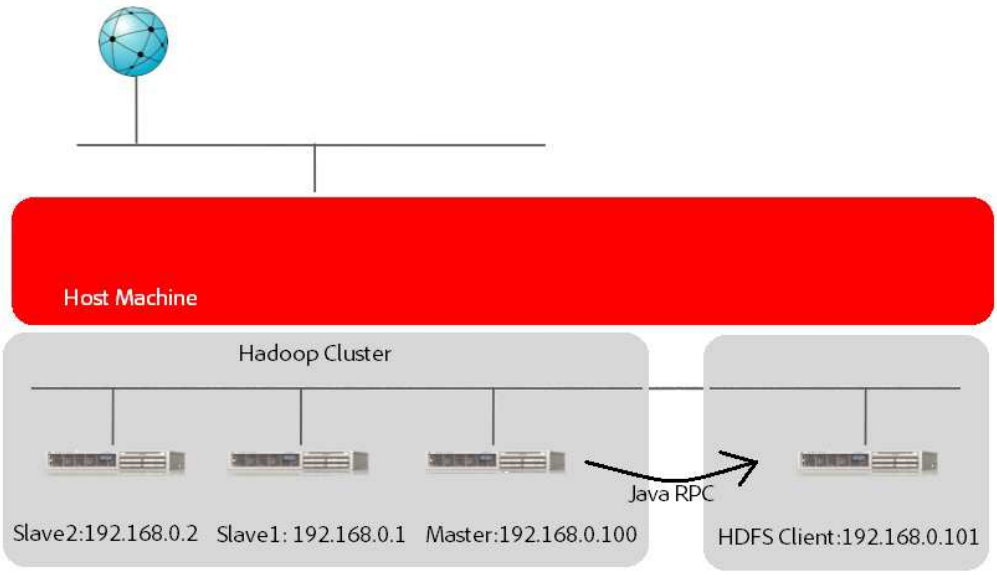

Fig. 8. Hadoop Cluster Experiment Diagram

File Size	1GB	3GB	5GB
1	33.47	100.12	169.29
2	30.98	96.59	167.32
3	33.34	101.05	166.90
4	29.41	102.11	169.23
5	33.61	101.82	165.59
Avg.	**32.14**	**100.34**	**167.67**

TABLE III
HDFS CLUSTER PERFORMANCE (IN SECONDS)

File Size	1GB	3GB	5GB
1	8.71	27.58	53.19
2	8.70	25.27	51.78
3	9.1	24.87	50.31
4	8.35	25.19	54.81
5	8.49	26.87	53.32
Avg.	**8.67**	**25.96**	**52.68**

TABLE IV
GRIDFTP PERFORMANCE (IN SECONDS)

2) ***Analysis****:* Based on the stats of the experiment, HDFS doesn't perform as well as GridFTP even though HDFS forms a cluster. Regardless the file size, GridFTP is around 70% faster than HDFS. Hadoop is installed on top of the server's file system. In order to access HDFS, a client needs to go through the server's hard disk first, and then get the file from HDFS. This extra hop could be the reason why it takes longer to read the file. On the other hand, GridFTP client merely reads the file from hard disk. The overhead is too big for a HDFS client to access HDFS by Java RPC. It leads us to believe that the HDFS access layer is the bottleneck for fast data transfer of Hadoop-based cloud storage.

D. GridFTP-over-HDFS

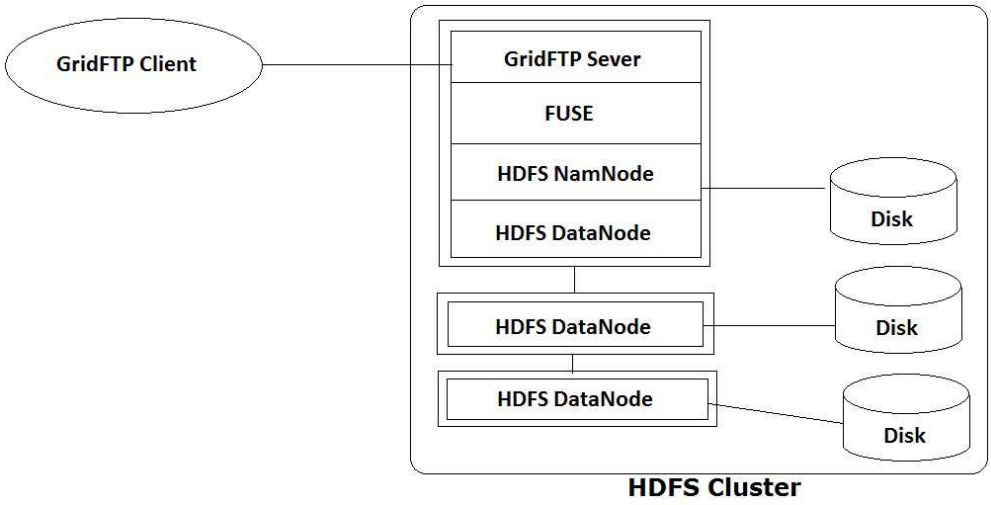

Fig. 9. GridFTP-over-HDFS ArchitectureDiagram

1) ***Design****:* Now we know that GridFTP outperforms HDFS, but HDFS has so many other features we would like to have as our storage system. For example, fault tolerance, integration with MapReduce paradigm, and process large datasets are very important when we have petabytes of file. Those are the functions we cannot get from GridFTP server. Is there a way to combine these two? FUSE (Filesystem in Userspace) would be the medium to make it possible.

HDFS cannot be directly mounted by the operating system. FUSE provides a POSIX-like interface to HDFS, which mounts the whole file system ”locally” so that user applications can access data as if it were local storage. It gave us the option to transfer files from HDFS using either FTP, GridFTP or other transferring methods instead of limiting to HDFS Java client. In other words, FUSE is the gateway between GridFTP client and HDFS.

2) ***Analysis****:* GridFTP-over-HDFS produces better results than HDFS client, but it is not as fast as using only GridFTP. We know that FUSE is the gateway to HDFS, so it creates overhead to read and write the file. However, the speed of GridFTP neutralizes FUSE overhead. This compromised solution will provide us a relative fast transfer rate, and in the meantime keeps all of the great features of HDFS. Moreover, X.509 authentication will be mandatory if we use GridFTP. It also solves the security weakness on HDFS. Thus, this has been a prevalent approach

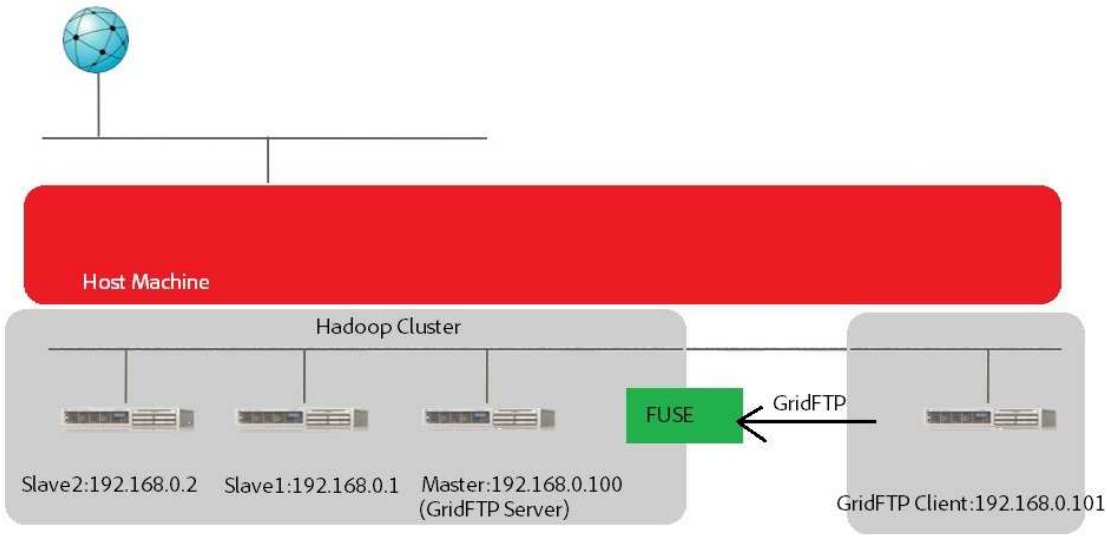

Fig. 10. GridFTP-over-HDFS Experiment Diagram

File Size	1GB	3GB	5GB
1	15.91	53.05	99.13
2	14.98	46.36	99.78
3	15.15	46.41	98.25
4	14.53	46.98	97.12
5	15.26	46.85	100.52
Avg.	**15.17**	**47.93**	**98.96**

TABLE V

GRIDFTP-OVER-HDFS PERFORMANCE (IN SECONDS)

for grid applications such as the Large Hadron Collider research project.

E. Secure HDFS v.s. HDFS

	Secure HDFS	**HDFS**
Security	Authenticate HDFS services and users by adding Kerberos	No Authentication on services and users

1) ***Design****:* GridFTP-over-HDFS doesn't authenticate HDFS. What if HDFS client is the only file transfer option? The current proposed solution would be adding Kerberos. It authenticates both HDFS service and client. However, we are still suspect communication time with KDC and delegation token would not have any impact on its performance. Therefore, we build another HDFS with a single node to compare the performance with secure HDFS.

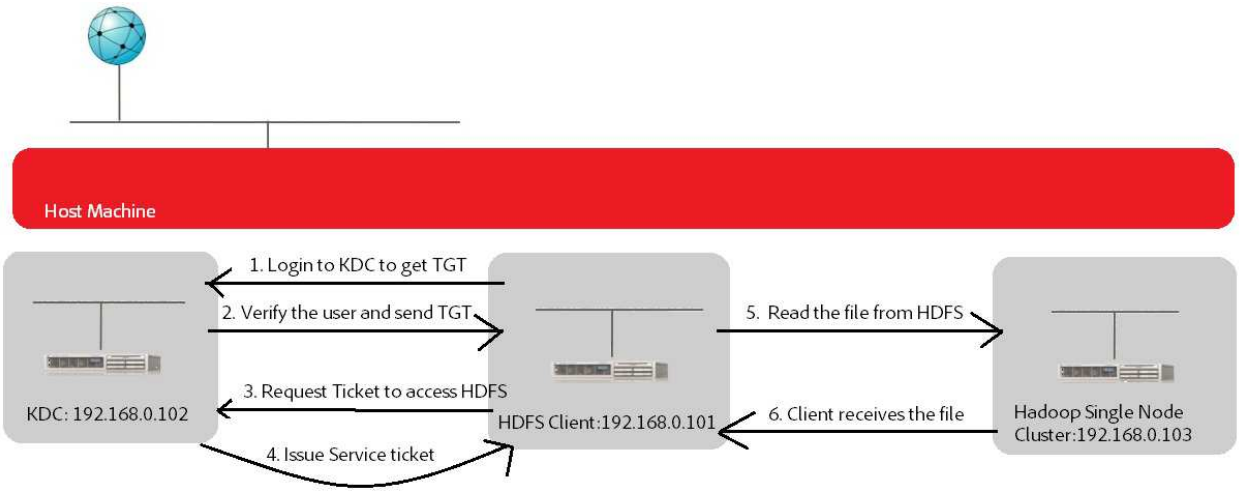

Fig. 11. Secure Hadoop Experiment Diagram

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
HDFS_Single	5	333.78	66.756	1.39688
HDFS_Secure	5	325.9	65.18	0.87225

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	6.20944	1	6.20944	5.472969817	0.047457	11.25862
Within Groups	9.07652	8	1.134565			
Total	15.28596	9				

Fig. 12. 1-way ANOVA Table of HDFS and HDFS Secure with significance level of 0.01 (1GB)

2) ***Analysis****:* Based on the experiment result, HDFS with Kerberos does not have a huge impact on the performance. Although the time of transfer for Secure HDFS is slightly slower than regular HDFS, the result is very close regardless the file size. We run 1-way ANOVA (Analysis

of Variance) test to check if there are scientific differences between them. We use significance level = 0.01 to compare these two, and ANOVA table in Figure 12 shows that F critical stats is larger than F stats for 1GB file. In addition, p-value 0.046 is bigger than the significance level 0.01. Statistical results leads us to retain null hypothesis. Therefore, performance differences between HDFS with and without security are statistically insignificant.

HDFS client connects with KDC initially to request Secure HDFS service. The connection between KDC and HDFS client can be seen by using ”tcpdump”. HDFS client basically asks for the service key of HDFS, and KDC issues it after KDC approves HDFS client’s identity. The experiment result shows that the authentication process of HDFS doesn’t slow down the file transfer process. When we run ANOVA test with larger file size (3GB and 5GB), the performance differences due to security become even less significant. Due to longer transfer time of larger file size, connection with KDC has less effect on the performance. We conclude that KDC is not the bottleneck of HDFS, and Kerberos has very little effects on file transfer of HDFS.

File Size	1GB	3GB	5GB
1	64.31	203.97	319.12
2	66.42	203.77	302.23
3	64.34	206.54	329.05
4	64.98	201.89	331.06
5	65.85	203.11	335.03
Avg.	**65.18**	**204.46**	**323.30**

TABLE VI

HDFS SINGLE-NODE PERFORMANCE (IN SECONDS)

File Size	1GB	3GB	5GB
1	66.24	210.52	319.64
2	68.74	203.29	315.59
3	66.6	208.4	347.53
4	66.6	210.16	320.38
5	65.6	206.35	329.75
Avg.	**66.76**	**207.75**	**326.58**

TABLE VII

SECURE HDFS PERFORMANCE (IN SECONDS)

In summary, we vary the size and conduct the experiment of different transferring protocol, but file size doesn’t appear to have huge impact on the performance results. We compare in total 6 different methods, and the trend is similar across the board. GridFTP is consistently the fastest transferring protocol, which is followed by either FTP or GridFTP-over-HDFS. The HDFS cluster is faster than single-node HDFS, and adding Kerberos to HDFS causes relatively small impact in terms of its performance. Based on the experiment, we propose a system ”GridFTP-Over-Secure HDFS” for secure and fast file transfer.

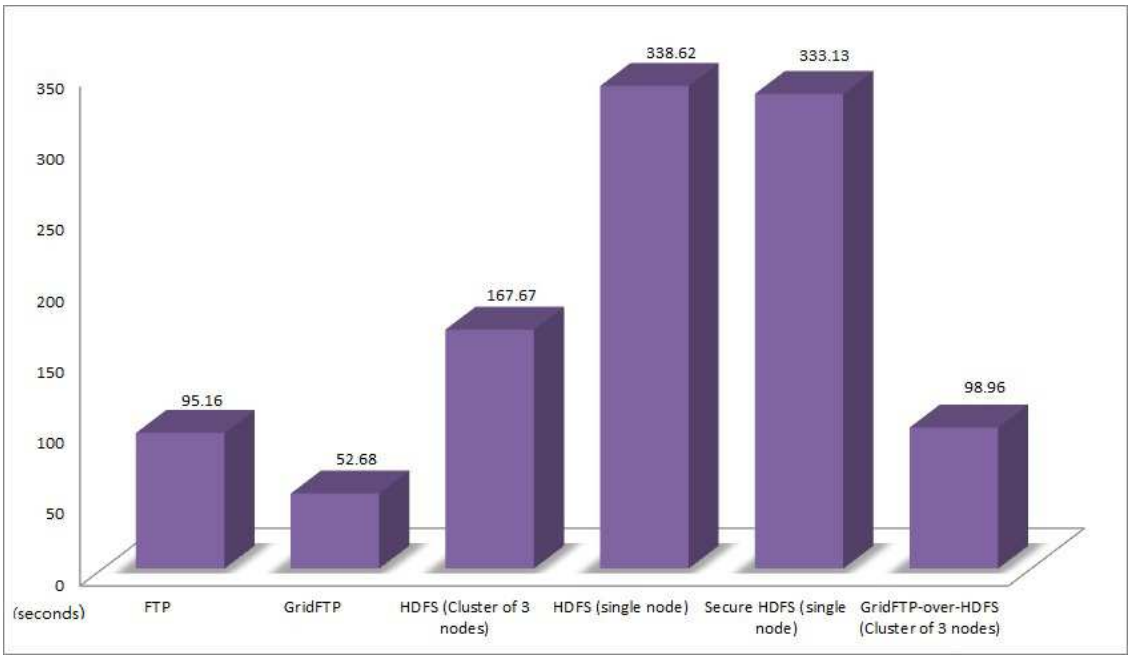

Fig. 13. 5GB Data transfer comparison graph (in seconds)

VI. GridFTP-Over-Secure HDFS System Design

According to the previous experiment results, we demonstrated two points

- GridFTP provides the best performance result
- Kerberos enhances HDFS security without much impact on the performance

In section V-D, we demonstrate GridFTP-Over-HDFS, and it shows decent throughput. Now we know that Kerberos won't affect performance, so a GridFTP-Over-Secure HDFS system could provide fast file transfer, and keep all the file at secure HDFS for distributed computing applications.

One possible solution is adding Kerberos on GridFTP-Over-HDFS, but this design is not feasible due to 2 limitations

- Apache HDFS Fuse module doesn't have Kerberos support yet
- Kerberos is not supported on Globus GridFTP due to divergence in the capabilities of GSI and Kerberos

Currently, GridFTP uses GSI for authentication and encryption. CA is a trusted third party (TTP) for GSI as Kerberos uses KDC as the authentication center. Although it is possible to replace GSI with Kerberos on GridFTP, secure HDFS still can't be accessed from GridFTP client without specifying keytab of HDFS on the command line URL.

Therefore, we implement another system without Fuse mounting secure HDFS. GridFTP-Over-Secure HDFS requires a user to operate through the command line from terminal. Here are the steps for a client to get the file from secure HDFS server.

(1) Client starts gsiftp and connects with the server ("test" is my server name)

```
$ export GLOBUS_LOCATION=/sandbox/globus/globus-5.0.5
$ source $GLOBUS_LOCATION/etc/globus-user-env.sh
$ myproxy-logon -s test
```

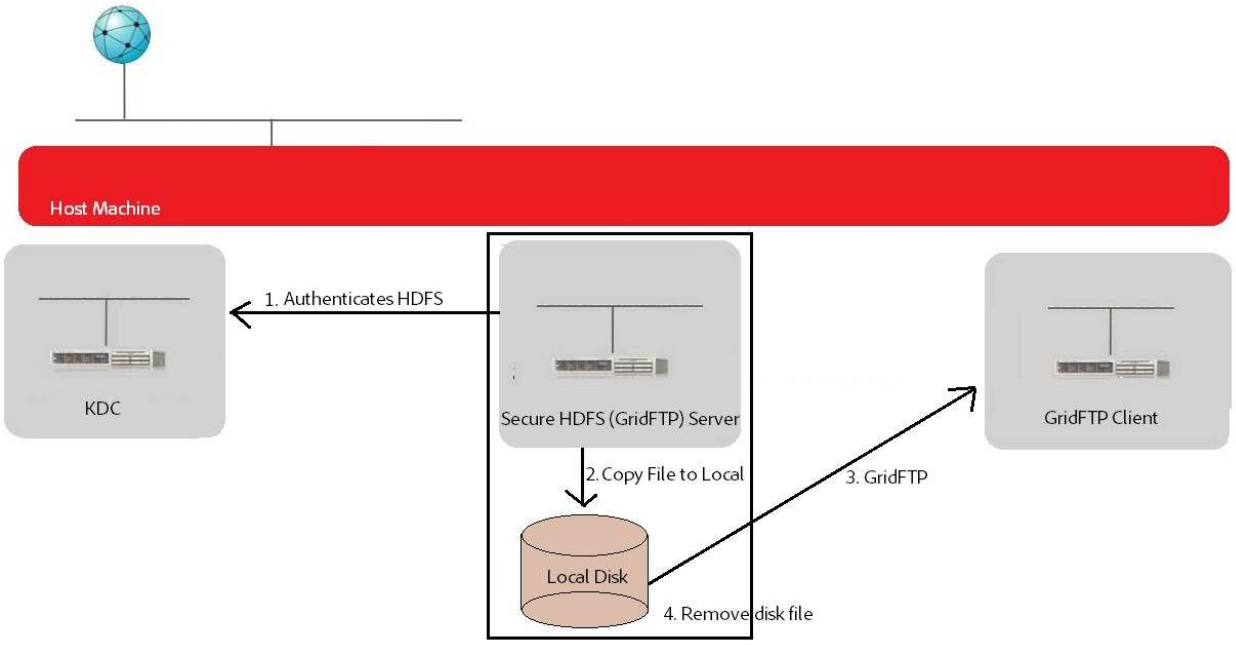

Fig. 14. GridFTP-Over-Secure HDFS Design: Copy a file to local then use GridFTP to transfer file

(2) Client SSH into the server

```
$ ssh yang@test
```

(3) Connect with KDC and enter the password correctly to recieve Kerberos key. The key is required to access the secure HDFS

```
kinit
```

(4) Search for files at HDFS

```
bin/hadoop dfs -ls /
```

(5) Calling a bash script **read_file** with 3 parameters. It specifies source path, client name and destination path

```
./read_file /hello testclient /home/yang/tmp
```

```
#!/bin/bash
/home/yang/Desktop/hadoop-1.0.2/bin/hadoop dfs -get $1   /Desktop/temp_file
#copy the file out of secure HDFS to the server's Desktop
globus-url-copy gsiftp://test/home/yang/Desktop/temp_file gsiftp://$2$3
#use GridFTP for file transfer between server and client
rm   /Desktop/temp_file
#remove the file on the local disk
```

(6) After file transfer completes, delete the Kerberos key

```
kdestroy
```

(7) Logout of server

```
exit
```

Script "read_file" copies the secure HDFS data to the server's local directory, and then starts file transfer between server and client with GridFTP. We run the same performance test with different file size, and the result is much better than secure HDFS. The script "read_file" executes Hadoop shell command "copyToLocal" on the first line, and apparently it doesn't cost much time to copy the file to local disk. In addition, we optimize transfer speed by using GridFTP since it is the fastest file transfer protocol in our experiment. After a client receives the file, the server's local copy is deleted immediately, because we do not want HDFS data exposed in any insecure environment.

File Size	1GB	3GB	5GB
1	32.04	91.91	153.54
2	30.91	98.67	157.75
3	30.4	91.78	158.85
4	28.85	98.05	151.29
5	31.81	98.39	150.88
Avg.	**30.8**	**95.76**	**154.46**

TABLE VIII
GRIDFTP-OVER-SECURE HDFS PERFORMANCE (IN SECONDS)

GridFTP-Over-Secure HDFS is the best solution comparing to the alternative system in the experiment. Firstly, FTP itself is inherently insecure, because it opens to packet sniffing and eavesdropping. Therefore, we replace it with GridFTP, because it is faster and contains security feature GSI for authentication and encryption to file transfers. However, GridFTP server alone is only good for simple file transfer. HDFS could run applications with multiple nodes involving thousands of terabytes, but lack of security features is its main issue. In addition to the security concern, the experiment also shows that HDFS cluster performs worse than GridFTP server in terms of file transfer. Thus, we build secure HDFS by adding Kerberos authentication, and use GridFTP to transfer HDFS file from server to client. In conclusion, GridFTP-Over-Secure HDFS outperforms HDFS and secure HDFS, so it is the best possible solution for HDFS file transfer.

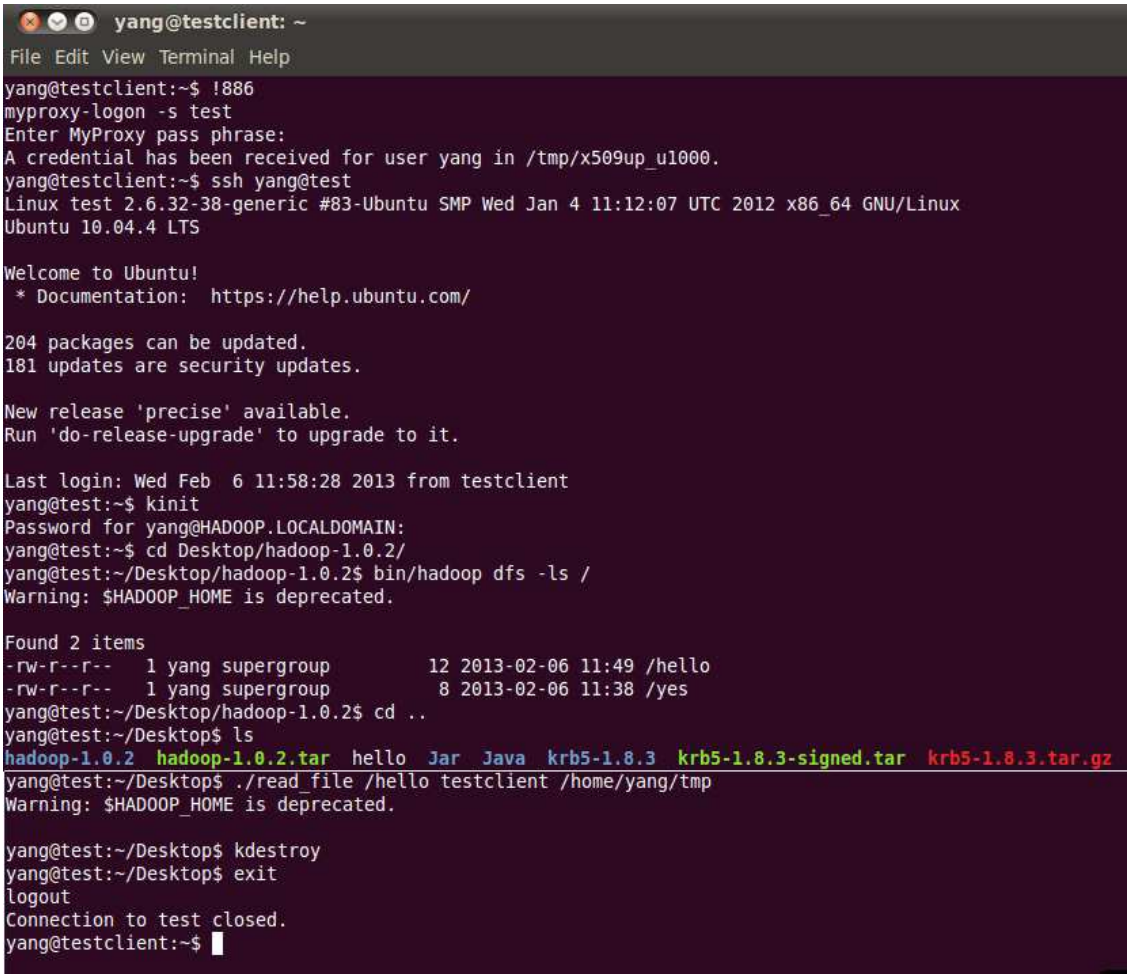

Fig. 15. Terminal User Interface to Transfer File from GridFTP-Over-Secure HDFS Server to Client

VII. Conclusion

From our experiment result, we find out that GridFTP performs better in file transfer comparing to HDFS. The overhead of accessing HDFS is quite big even HDFS cluster system cannot nullify. However, with FUSE we could have GridFTP client stores content directly into HDFS. It allows data from GridFTP serves to be directly stored into HDFS instead of first copying the data locally and then uploading it into HDFS. This will greatly simplify data being pulled from GridFTP Servers to HDFS. In addition to fast data transfer, GridFTP could also solve the issues of managing multiple cloud service accounts on the internet. Cloud storage systems today all have their own file upload and download methods. GridFTP is based on FTP, so it is compatible with most of the cloud service. We can use GridFTP to move or copy files between any accounts.

Cloud service offers more dynamic features FTP server does not have, but our experiment also shows that FTP is faster than one of the cloud storage platforms (HDFS). HDFS still has some flaws to become a reliable cloud storage system. The authentication issue is solved partially by adding Kerberos, but there are many security risks existing, including unencrypted data on the network. With more work are being done in this field, the proposed design (GridFTP-Over-Secure HDFS) is currently a great solution for a fast and reliable cloud based storage system.

REFERENCES

[1] SearchCloudComputing. (2012, Mar. 19) Hadoop. [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/Hadoop

[2] D. Borthakur. (2008) Hadoop 0.20 documentation – HDFS architecture. The Apache Software Foundation.

[3] K. Shvachko, "The Hadoop Distributed File System," in *IEEE Symp. on Mass Storage Systems and Technologies*, 2010.

[4] Yahoo! Inc. (2012, May 7) Hadoop tutorial. Yahoo! Developer Network. [Online]. Available: http://developer.yahoo.com/hadoop/tutorial/module2.html#perms

[5] R. Phulari. (2010, Mar. 22) Security in Hadoop, part - 1. Big data and etc. [Online]. Available: http://bigdata.wordpress.com/2010/03/22/security-in-hadoop-part-1/

[6] Q. Shien, "Security architecture of private storage cloud based on HDFS," in *IEEE Computer Society, 26th Int. Conf. on Advanced Information Networking and Applications Workshops*, 2012.

[7] MIT. (2012) Kerberos: The network authentication protocol. MIT Kerberos. [Online]. Available: http://web.mit.edu/kerberos/dist/historic.html

[8] H. Jacob. (2012, May 6) Up-armoring the elephant. [Online]. Available: http://www.slideshare.net/blueboxtraveler/uparmoning-the-elephant-adding-kerberosbased-security-to-hadoop

[9] N. Chary. (2012, May 6) Security implementation in Hadoop. [Online]. Available: http://search.iiit.ac.in/cloud/presentations/28.pdf

[10] B. Alcock, "Data management and transfer in high performance computation grid environments," *Globus.org Publications*, 2002.

[11] G. Attebury, "Hadoop Distributed File System for the Grid," in *IEEE Nuclear Science Symp. Conf. Rec.*, 2009.

[12] B. Andrew. (2010) Hadoop security design just add kerberos? really? iSEC PARTNERS. [Online]. Available: http://media.blackhat.com/bh-us-10/whitepapers/Becherer/BlackHat-USA-2010-Becherer-Andrew-Hadoop-Security-wp.pdf