An Email based Offline Download Manager for Large Distributed File System using Hadoop Map Reduce Framework

Pradeep H K, Rohitaksha K, Abhilash C B
Assistant Professor, Computer Science & Engineering,
JSS Academy of Technology Bangalore, Karnataka, India

Abstract **-** People are using P2P (Peer to Peer) network for sharing and transferring digital content contains video, audio, and any other data files over the internet from different part of the world. The general peer to peer file sharing protocols was designed to work optimally in the case that all the peers have an end node on the internet. The end peers should capable of delivering the contents in proper way with limited time constraint. To solve this problem, a method is proposed a new approach that will be implemented using email's as a medium for data transfer and load balancing with the help of Hadoop-MapReduce framework and moreover if we use systems like Gmail and Yahoo then most of the mails would be transferred internally and with more efficiency, thus improving the overall efficiency of the internet. MapReduce is a framework which is pioneered by Goggle for distributed programming. It includes user specified Map and Reduce functions which process inputs in the form of key/value pairs. Along with the MapReduce paradigm, Hadoop also implements HDFS which is known as Hadoop distributed file system.

Keywords **-** *peer to peer (P2P), Hadoop, MapReduce, HDFS.*

1. INTRODUCTION

P2P file sharing makes up the bulk of internet traffic nowadays. But most of them are inefficient in different ways, and rely on the presence of connectable hosts (end node on the internet). Currently there are lots of P2P based applications like eDonkey, SoulSeek, DC++, Lime Wire, eMule and Bittorrent etc. But all of them work within TCP/IP. Because of this all system suffer from the same disadvantages as enumerated:

Reachability Problem occurs if 2 nodes are behind a proxy or firewall or any other NAT device they cannot contact each other since they don't have a reachable global IP Address. Due to the shortages of IP addresses more and more service providers are shifting over to NAT1. Currently no method exists for providing reachability to users behind proxies, but attempts have been made to establish connectivity between hosts behind NAT. Studies in have shown that NAT Traversal techniques give efficiencies of about 82% in UDP and 64% in TCP. But these methods require the use of a meditating server [1].

P2P blocking in most of the networks disallow or ban P2P on their networks due to heavy traffic. Due to this many users cannot use file sharing with the outside world. Most proxy servers only allow outbound access to a few service ports (like HTTP, SMTP, POP3, Telnet and SSH etc.)

Low Upload speed of clients in most broadband connections that is the download speed is usually much higher than the upload speed. Most general broadband connections in the world provide ADSL connections which are symmetric in nature [1].

Due to the shortage of IPv4 addresses and IPv6 compatibility problem it's not possible to make all peers reachable and since many OS's and programs still lack support for IPv6. P2P applications haven't made the switch to IPv6 yet. Thus in the current situation it's not possible to have all reachable clients in a P2P network [1].

More Load on Each Peer since Each Peer may get a lot of load since it requires a lot of time to send the requested mail. This imposes a more idle time for the client [1].

The above stated problems motivated us to formulate a new p2p file transfer protocol which can overcome all the stated drawbacks of the available p2p file transfer protocols. Load balancing problem stated above is overcome here by using Hadoop-MapReduce Framework.

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real world tasks. The MapReduce technique in cloud can be applied here[2].Users specify the computation in terms of map and reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of networks and disks. Google and Hadoop both provide

MapReduce runtimes with fault tolerance and dynamic flexibility support. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. The implementation of MapReduce runs on a large cluster of commodity machines. In a typical MapReduce computation processes many terabytes of data on thousands of machines are used. Implementers find the system easy to use and hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed frequently [3].

2. RELATED WORK

Currently No P2P protocols exist that work at the email level. The only software remotely related to email is Pando [9]. It use email to transfer the metadata only and all the other data transfer occurs though TCP/UDP. Moreover, the user has to manually open the inbox, download the metafile and start it in the application Pando. Then it starts transferring the files from the Internet using TCP. Thus, we would not call this an email P2P file sharing application.

Bit Torrent is the most widely used P2P protocol on the internet. Bit Torrent works by establishing end to end connections between the hosts, and using them to transfer files. There are two versions of the protocol, one depends on the presence of a tracker to communicate with the peers, other doesn't require the presence of trackers, and it is based on DHT (Distributed Hash Table). Bit Torrent [6] in itself is only a file-downloading protocol. In Bit Torrent, files are split up into chunks (on the order of a thousand per file), and the downloaders of a file barter for chunks of it by uploading and downloading them in a tit-for-tat-like manner to prevent parasitic behavior. Each peer is responsible for maximizing its own download rate by contacting suitable peers, and peers with high upload rates will with high probability also be able to download with high speeds. When a peer has finished downloading a file, it may become a seed by staying online for a while and sharing the file for free, i.e., without bartering.

Direct Connect protocol is based on the concept of hubs clients and a super hub. Peers connect to the hubs. The hub servers as a connecting point for all the peers. Peers can view the files shared by other peers, and transfer them. Advanced Direct Connect can be considered a successor protocol.ADC is structured around clients that connect to a central hub, where the clients (users) can chat and download files from other clients (users). The hub provides routing between clients for chat, searches and requests for connections. The actual file transfers are between clients.

Gnutella [5] is a fully distributed file sharing protocol. In this protocol each Gnutella client is connected to at least one client in the network After that the Gnutella clients asks for a list of peers from the other client. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model. In this model, every client is a server, and vice versa. These so-called Gnutella servants perform tasks normally associated with both clients and servers. They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servants, check for matches against their local data set, and respond with applicable results. Due to its distributed nature, a network of servants that implements the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servants goes offline.

Even searching is done in a distributed way, clients disseminate the search query to the nodes that are directly connected to them.

3. BIG DATA

Big Data is nothing but bigger data chunks—imagine song preferences of 10 million people from 50 different nationalities, divided according to the music genre, in multiple servers—that's a lot of decimals—categorized into sets of data based on data similarities. These data sets are then analysed to gain insights and information.

Big Data sizes are a constantly moving target, as of 2012 ranging from a few dozen Terabytes (1 terabyte = 1024 Gigabytes) to many Petabytes (1 petabyte = 1024 Terabytes) of data in a single data set.

Analysing Big Data [8] involves massive numbers of parallel software running on tens, hundreds, or even thousands of servers. Big data analytics is done with the software tools commonly used as part of advanced analytics disciplines such as predictive analytics and data mining. But the unstructured data sources used for Big Data analytics may not fit within traditional data warehouses. A traditional method of data processing and analysis was never built to accommodate the complexities of size, diversity and movement, making it difficult to process Big Data. As a result, a new class of technology has emerged and is being used in many big data analytics environments. Clever IT organizations working in this realm quickly realized the limitations of traditional approaches—economics, operations, lack of agility, etc. —and decided to provide Big Data users with something better and more cost-effective.

The technologies associated with big data analytics include NoSQL databases, Hadoop MapReduce framework.

Analysis of Big Data requires clusters of servers, running distributed analysis backed by complex programming models. Software like Hadoop is almost completely modular, which means that you can swap out almost any of its components for a different software tool. That makes the architecture incredibly flexible, as well as robust and efficient

4. HADOOP

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. An important characteristic of Hadoop [7] is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data. Hadoop's distributed compute framework called MapReduce, exploits the distributed storage architecture of Hadoop's distributed file system HDFS to deliver scalable, reliable parallel processing services for arbitrary algorithms. The shuffle phase of Hadoop's MapReduce computation involves movement of intermediate data from Mappers to Reducers.

The Apache Hadoop framework is composed of the following modules:

Hadoop Common – contains libraries and utilities needed by other Hadoop modules

Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster. The Hadoop Distributed File System (HDFS) [4] is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size.

Hadoop YARN – a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.

Hadoop MapReduce – a programming model for large scale data processing.

Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google MapReduce and Google File System (GFS) papers.

5. MAPREDUCE

'MapReduce' is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). Computational processing can occur on data stored either in a file system (unstructured) or in a database (structured). MapReduce can take advantage of locality of data, processing data on or near the storage assets to decrease transmission of data.

"Map" step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

"Reduce" step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.

The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g., hash (key) mod R). The number of partitions (R) and the partitioning function are specified by the user [3].

Figure 1 shows the overall flow of a MapReduce operation in our implementation. When the user program calls the MapReduce function, the following sequence of actions occurs (the numbered labels in Figure 1 correspond to the numbers in the list below):

1. The MapReduce library in the user program first splits the input files into M pieces (controllable by the user via an optional parameter). It then starts up many copies of the program on a cluster of machines.

2. One of the copies of the program is special the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.

3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-designed Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.

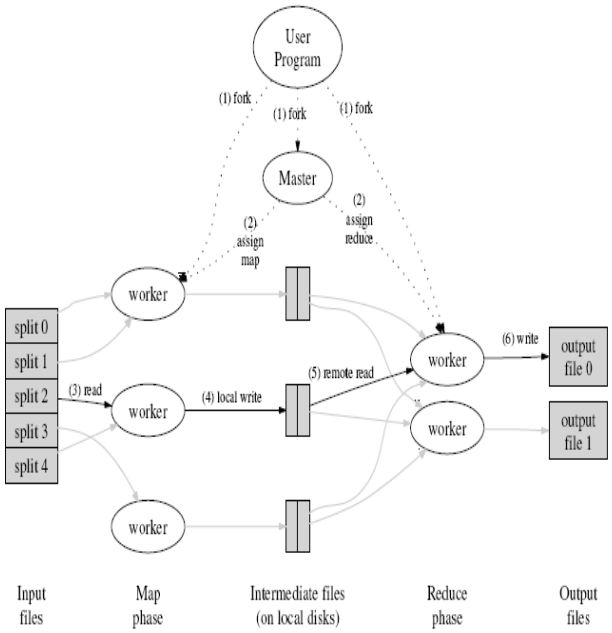

Fig-1: MapReduce

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.

6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

After successful completion, the output of the MapReduce execution is available in the R output files (one per reduce task, with file names as specified by the user).Typically, users do not need to combine these R output files into one file they often pass these files as input to another MapReduce call, or use them from another distributed application that is able to deal with input that is partitioned into multiple files.

6. PROPOSED APPROACH

The proposed system makes use of Hadoop Map Reduce framework for solving the problem of peer-reachability and load balancing of existing systems.
The Hadoop framework makes use of the MapReduce feature to pass the incoming request splitting the tasks to the multiple task executors and getting the combined report of task completion status.
The System involves the 3 Parties

1. The Client
2. The Hadoop Mapper
3. The Task Executors

Since, email providers usually set limits on the maximum size of email that can be sent, so to get over this limitation the file should be broken pieces. Moreover, breaking it into pieces allows partial piecewise downloading for a file. So the protocol basically deals with the various steps involved in getting these pieces to all the hosts.
The main functionalities of the Client side application are, sending the request another, downloading the mail from the Gmail server and integrating them to become a single piece.
Requesting for a file is the First main part in the client side. When the client sends the request he has to send the Gmail Id with password and the metadata file to the mapper or the Hadoop server. The Metadata file maintains all the information like name of the file, number of pieces, and name of each piece of the file.
The Client side Mail Receiving application will check the mail of the client each minute. When the file pieces are received the save attachments function will downloads the files from the mail and saves the divided file chunks to the admin folder. Once all unique pieces of the file from the mail are downloaded completely then the integration will be done by sequentially reading each file piece and writing to the final file.
The Client side Mail checking application has to check the mail for the files, as Hadoop will assign the files to more than one cluster so more than one copy of the same file will arrive in Gmail. The Client application has to take only one piece from each similar files, So there will be only one copy of a file in client system.

The main task of the Hadoop Mapper is to schedule the tasks to the Task Executers. The Hadoop Mapper will have the list of files with them that are already divided and also have the number of divided files. Once the request for a file is received by the Hadoop Mapper then a check is performed to see if the client requested file is present in the Hadoop Mapper.If the files are present then they will forward the request to the clusters. Here the mapper checks for the load balance in the clusters. If the clusters are busy then the Hadoop will move to other clusters which are free and can perform the work faster. Defending upon the load present in the different clusters they will allocate the task to them.

The Hadoop Mapper will also check the file size and the network load of the task executer if the load on the single network is more than the other then the file from the lesser loaded task executer will be transferred faster.

The Hadoop will map the data in more than one cluster because if one cluster fails to transfer the data to the proper destination or correct email Id because of the network failure or more load in the network at that time other clusters can pass the same message over the network to the client thus saving the time or loss of data from one cluster. Sometimes we will get the file from more than cluster.

The Task Executers main function is to forward the requested file to the requested client Email-id. The task executer will get the forwarded request from the Hadoop Mapper. The Hadoop Mapper will send the request only to a few task executers those who have the pieces of the file. The task executer will forward the file piece to the client Email-id. The Task Executor will have the split files thus maintaining the load across the network. They will transfer these divided files one by one to the client as they will finish mapping and reducing of each piece of the file. Once 100% completion is displayed in the map and reduce function then the particular piece of the file which has been mapped is sent to the client.

The Requirements of the system are

1. In this system the user can send the mail or any file at any time he wants.
2. The Gmail server should receive this file when the user sends it.
3. The proposed system needs to reduce the polling time wasted while waiting to receive the file.
4. The Hadoop cluster must be able to notify the receiver of the file as soon as the other user sends the file.
5. The system must work in the absence of the other user.

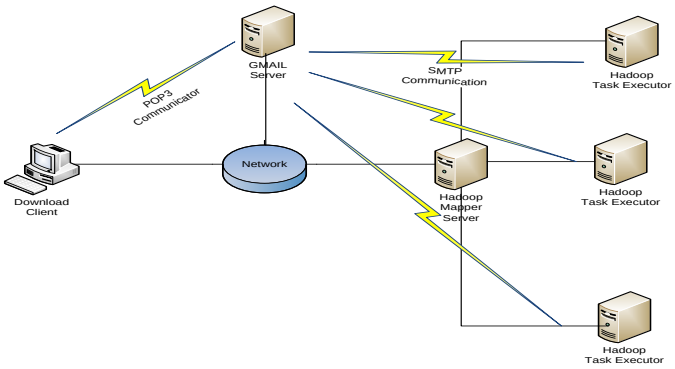

Fig-2: System Architecture

7. IMPLEMENTATION

SMTP is the de facto standard for sending messages and IMAP/POP3 can be used for retrieval.

The implementation can be divided in to four distinct processes

- File splitter process.
- Data Integrator process.
- Job scheduler manager.
- Email polling process.

File splitter process include

- Divide the file in to 1MB pieces.
- Store pieces to file separately.
- Store pieces in Hadoop clusters.

Data Integrator process includes

- Read downloaded file chunks.
- Integrate the pieces sequentially.
- Store Data to final file.

Job scheduler manager includes

- Read Email-Id from Job request.
- Create Mail with Attachment Data.
- Send Email.

Email polling Manager includes

- Connect to Gmail.
- Check Data Availability.
- Download file content.

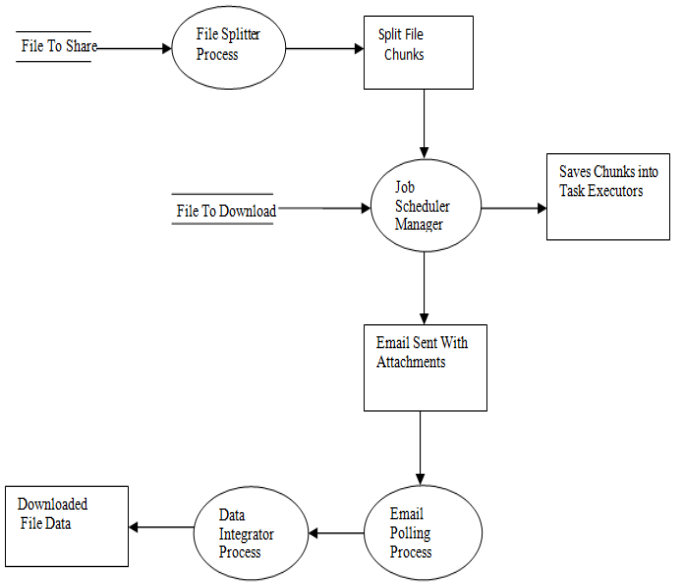

Fig-3: Data flow diagram of the system

The system flow includes:

- File Splitter Process is concerned with splitting the given file into pieces. The files split will be stored in hadoop file system.
- Once that file is split and stored in hadoop file system, the details of the split file chunks will be sent to the hadoop file scheduler.
- Job scheduler manager is concerned with sending the split files to the client from the clusters upon receiving the Email-id of the client.
- Email polling process checks whether mail has been received.
- Once all the piece of the file is received by the client. Data Integrator Process will be used to download the pieces of files from the mail server of the client and integrating those split files to get the requested file.

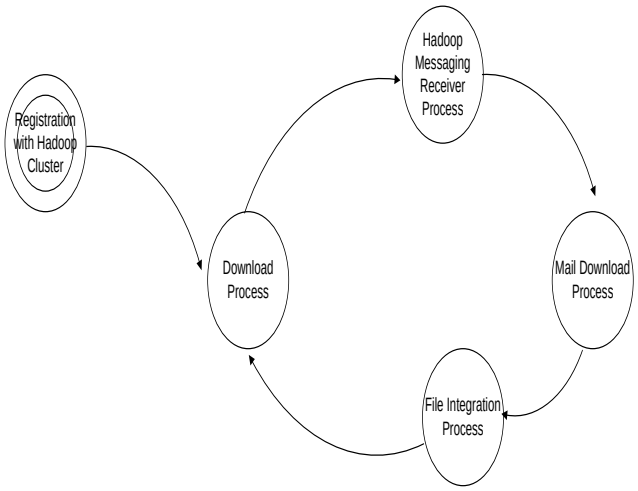

Fig-4: Data flow diagram of client

Client will send the file download request to the scheduler along with his Email-id and password. The scheduler will map the files to the appropriate clusters which will forward the mail. The Hadoop message receiving process will check the mail server of the client once the pieces arrive, the Mail download process will download the pieces on to the client system and the File integration process will integrate them to a single file.

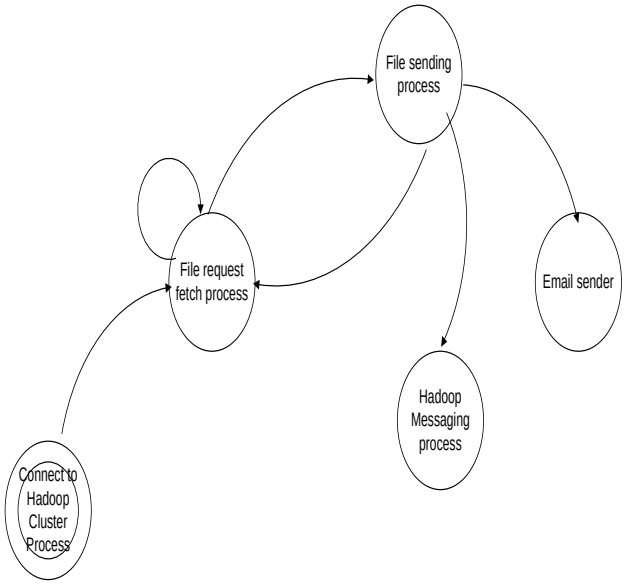

Fig-5: Data flow diagram of server

The file request fetch process will fetch the request for the file and map them to appropriate clusters. The File sending process will contact the Hadoop messaging process to obtain the Email-id and password. Email sender process is used to forward the file to the mail server.

As a result of implementing these modules, we were able to come up with our application and here are a few snapshots of the same.

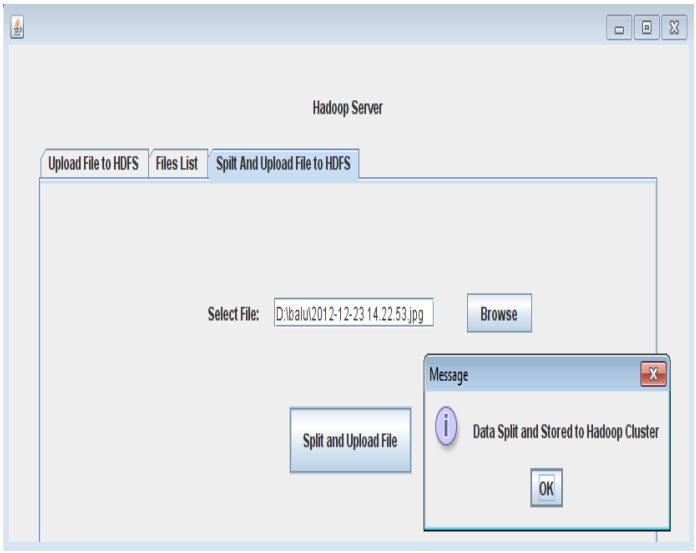

Fig-6: Hadoop server

Fig-6 shows the snapshot of Hadoop server where a file is split in to 1mb pieces and stored in Hadoop file system.

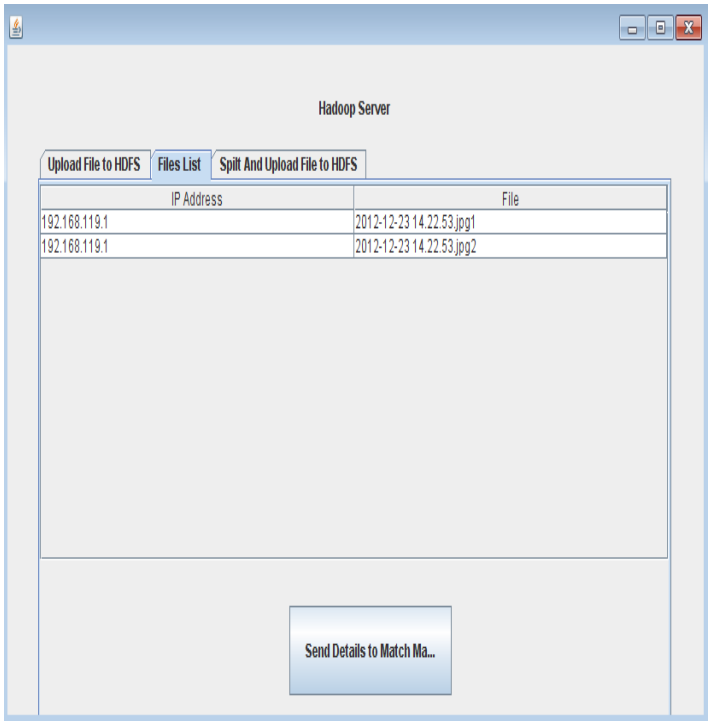

Fig-7: File list of Hadoop server

Fig-7 shows the snapshot of File list of Hadoop server. These are the details of the files present in HDFS. Send Details to Match maker button sends the files details to the scheduler.

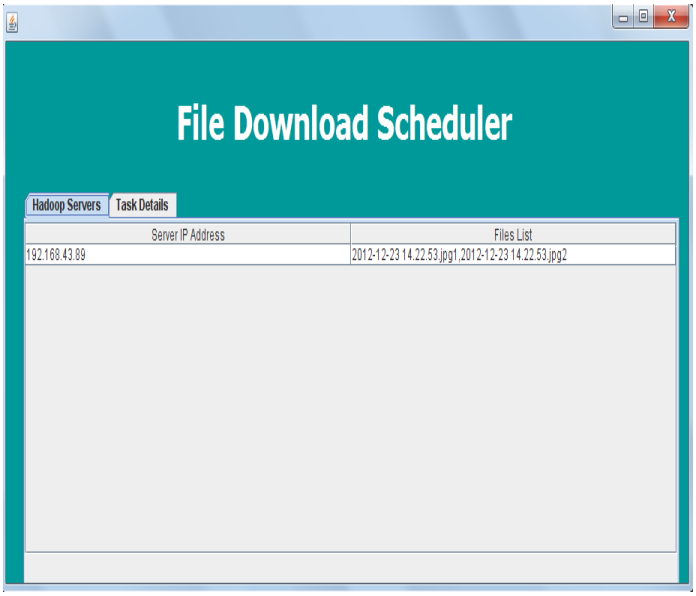

Fig-8: File scheduler

Fig-8 shows the snapshot of file scheduler after receiving the file details from Hadoop server.

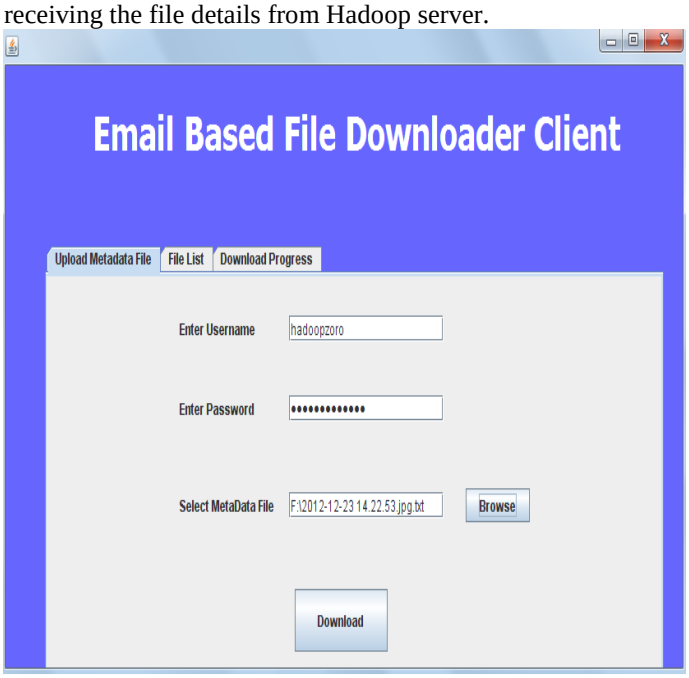

Fig-9: client application

Fig-9 shows the snapshot of client requesting for the file by specifying the metadata file.

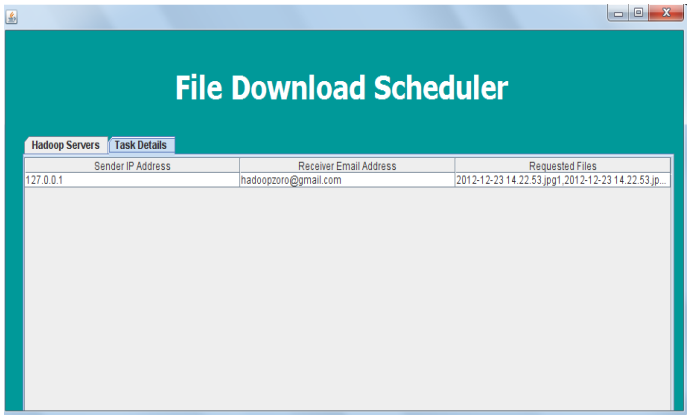

Fig-10: Task details of scheduler

Fig-10 shows the snapshot of File scheduler after receiving the request from the client.

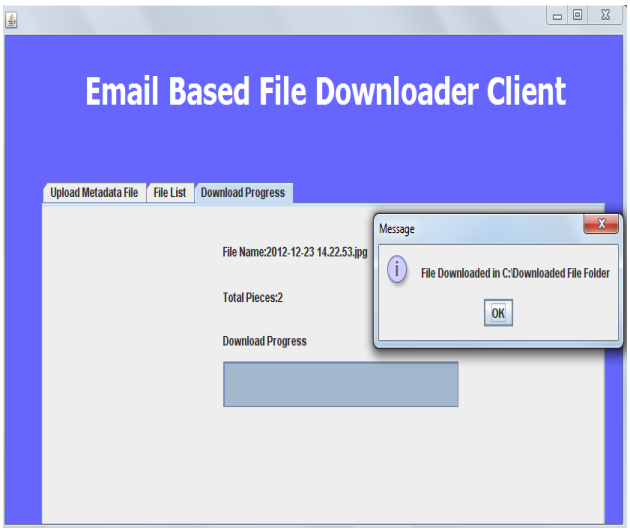

Fig-11: client application

Fig-11 shows the snapshot of client after downloading of split files and integration of the split files to a single file is complete.

8. CONCLUSION

In this paper we have proposed a peer to peer file sharing protocol that will be implemented using email as a medium of data transfer which would be a huge improvement over existing p2p networks since every node would be available, and would be possible to send a file to multiple users without uploading it multiple times, since we are using Gmail server the overall efficiency of the internet is improved. All P2P file sharing protocols work when the peers are connectible. Due to this, the load is unevenly distributed between the connectible, while connectible users suffer from too many uploads. In the case that all the peers are not connectible it is not possible to make use of peer to peer at all. In this paper, we present an entirely new p2p protocol which takes care of the deficiency of Load balancing by using Hadoop MapReduce Framework.

REFERENCES

[1] MailZoro Email Based P2P File Sharing. Ajit D Dhiwal1, Sudip Gautam2, Akshay K Singh3, Vijay K. Chaurasiya 4 IIITAllahabad, India, 211011.

[2] A Scalable Two-Phase Top-Down Specialization Approach for DataAnonymization using MapReduce on Cloud
Xuyun Zhang, Laurence T. Yang, Senior Member, IEEE, Chang Liu, Jinjun Chen, Member, IEEE.

[3] MapReduce: Simplified Data Processing on Large Clusters Jeffrey Dean and Sanjay Ghemawat.

[4] The Hadoop Distributed File System Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler Yahoo!Sunnyvale, California USA.

[5] The Gnutella Protocol Specification v0.4 Clip2 Distributed Search Services.

[6] THE BITTORRENT P2P FILE-SHARING SYSTEM: MEASUREMENTS AND ANALYSIS J.A. Pouwelse, P. Garbacki, D.H.J. Epema, H.J. Sips Department of Computer Science, Delft University of Technology, the Netherlands.

[7] Parallel & distributed processing (ipdps), 2010 ieee international symposium on mapreduce programming with apache hadoop bhandarkar, m. ; yahoo! Inc., hadoop solutions architect

[8] Big Data analytics Singh, S. ; Bus. Analytics Div., IBM India Software Lab. (ISL), Pune, India ; Singh, N.