VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE

Course Work

Distributed File Systems

Done by:
Adomas Bazinys signature

Supervisor:
Doc. Linas Bukauskas

Vilnius
2019

Contents

Abstract

Goal of Distributed File System is to decentralize a network and make it more e cient by increasing files availability. Distributed File Systems were invented to make data exchanging more faster and comfortable.

This work will help to get familiar with possible types and architecture of Distributed File System. This work will focus on InterPlanetary File System and it will be analyzed in more detail. There will be also upload/download tests in this work in order to measure performance of IPFS system.

Santrauka

Paskirstytų Failų Sistemos

Paskirstytųjų failų sistemų tikslas yra pilnai decentralizuoti tinklą ir padaryti jį veiksminegsniu padidinant failų prieinamumą. Šios sistemos buvo sukurtos padaryti duomenų keitimąsi greitesniu ir patogesniu.

Šis darbas padės susipažinti su egzistuojančiais paskirstytųjų failų sistemų tipais ir jų architektūra. Darbas orientuotas į IPFS (Eng. InterPlanetary File System) failų sistemą, kuri bus išanalizuota detaliau. Taip pat darbe bus pateiktas parsiuntimo - išsiuntimo testų rezultatai, kurių tikslas ištestuoti sistemos našumą.

List of Abbrevations

DFS - Distributed File System.

IPFS - InterPlanetary File System.

DHT - Distributed Hash Table.

P2P - Peer To Peer.

DHT - Distributed Hash Table.

DSHT - Distributed Sloppy Hash Table.

FTP - File Transfer Protocol.

Introduction

In the 1980s people who wanted to share data between physical machines did it slowly and uncomfortably. A method that has been used for that is known as „Sneakernet“. This is a slang term based on transferring of data and files between physical machines through devices like flash drives, hard drives, optical disks, which were physically transported between machines on foot. It was very ine cient and time consuming [6]. As response to this situation, FTP system were developed for that and became popular very fast. FTP made the situation much easier, it was also far from comfortable, data still had to be replicated twice: first from the computer to the server, then from the server to the target computer [5]. J. C. R. Licklider in 1962 has generated an idea to create a network, where all computers are able to exchange data between each other. His idea was remembered. His idea is a possible functionality of Distributed File System [11].

Distributed File System is a file-service system that helps to manage data files across multiple nodes or more precisely computers. The main di erence between File System and Distributed File System is that on File System data is stored locally, on Distributed File System instead of storing data on a single machine, data is stored on the clusters which is a band of computers connected to each other. If you try to use data from anyone from the machines of the cluster, it will be like a data is stored on the machine you use [13].

After solving problem of inconvenient data transfer, the next stage is to get best performance capabilities. People want pages, images, videos or other important files to load instantly and we want them in high quality.

Goal of the work: Analyze IPFS DFS, its operating principles, advantages, test performance of IPFS by creating a few IPFS nodes and measuring performance of downloading and uploading some data through IPFS system.

Tasks:

- Define and analyze DFSs overall.
- Define main components of IPFS system.
- Analyze IPFS system architecturally.
- Launch IPFS and measure performance.

1 Distributed File Systems

Distributed File System is a system on mechanism by which we can implement a common file system on di erent independent systems or computers. The goal of Distributed File System is to have maximum transparency and high availability properties. Reliability of DFS is achieved by making a lot of certain file copies through multiple servers ensuring that a copy of file will be accessed from a user if needed [10].

1.1 Fault tolerance

An error in a computer or a communication could cause a troubles in file transferring. This property also a ects availability of the distributed file system and also could reduce a consistency level of DFS when several users try to download a certain file [10].

1.2 Transparency

DFS should not only provide high level of transparency, but also to hide details related to the distributed environment. The types of transparency specific to a DFS are shortly explained below [12].

- **Access Transparency** - content could be stored locally or distributed, with same operations.
- **Location transparency** - to access object there is no need to know their location.
- **Concurrent transparency** - concurrent control without a ecting users or applications.
- **Replication Transparency** - maintains consistency of multiple backup copies without user noticing.
- **Failure Transparency** - normal users do not have to know about failures in the system, only admins should be informed know about them.
- **Migration transparency** - movement of objects must be invisible to the users of a system.
- **Performance transparency** - if reconfiguration is needed then it do not have to influence an instant performance of a system.
- **Scaling transparency** - system and apps can expand in scale without changing system structure or application programs.

1.3 Replication

Replication of documents on various machines in a DFS helps to enhance file accessibility. It has also a huge impact on performance: choosing a nearby copy of a certain file reduces a file service time. The accessibility of one replica is not influenced by the accessibility of other replicas of the file in a network. It is desirable to hide the details of replication from users. Mapping a replicated file name to a particular replica is the task of the naming scheme [10].

1.4 Synchronization

Purpose of synchronization is to make sure that when file is updated, then all replicas of that data is updated in Distributed File Systems. Replica synchronization ensures that the file is updated as many times as it is altered. [16]. There are three methods for synchronization [4]:

- Synchronous method blocks all requests of data until all copies of file is not updated.
- Asynchronous method is opposite to synchronous because it allows to modify a data even if all the copies are not update.
- The semi-asynchronous method blocks requests until certain copies are updated, but not all. This method limits access to old data in the system.

1.5 Naming

Names are all sources of DFS that consist of computers, services, users, and remote objects. A DFS must make sure that objects are named orderly and clear [10].

2 Structure of Distributed File System

DFSs is designed to use less hardware resources as possible. Thousands of hardware resources are used to achieve this characteristic by combining their resources to one big file share system. DFSs provide a lot of advantages in file management. Design of DFS is created to provide file exchange services to clients. Clients are able to manipulate with files by reading, creating, deleting, writing files. System which is able to perform such an operations could be considered as distributed file system [10]. This section will be based on information from Tannenbaums book.

There are three structures that DFS can be based on [15]:

- Server-Client based structures.
- Cluster-based structures.
- Symmetric structures.

2.1 Server-Client based structures

The first figure is remote access model (see 1 Fig.) representing Server-Client based structure, where an interface with various file operations is provided by a client. File sharing operations are carried out using that interface. The server is responsible for giving an answer to those requests. In such a model file is available only on server: clients send commands to server and get work done [10].

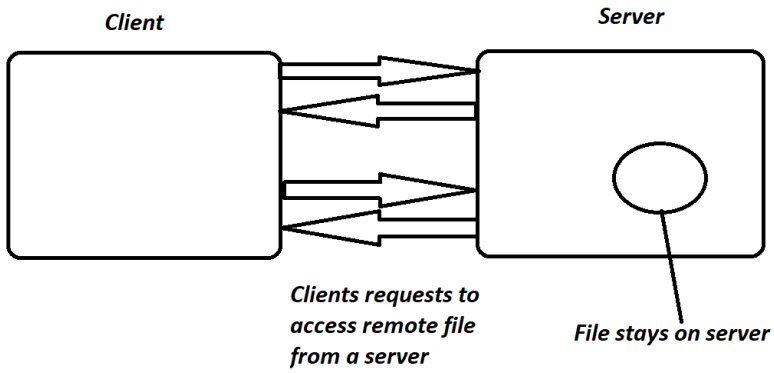

Figure 1. The remote access model.

The second figure is Upload/Download model (see 2 Fig.) representing Server-Client based structure. Feature that separates Upload/Download and Remote-Access is that Upload/Download model download the file that client will work on, and will download file to the local file system. In this model client downloads file, works on it, and writes it back on the server following a simple and good performance [10].

These both models are of server-client based structure and have a certain benefits. The Upload/Download model is not so time consuming because it requires to send only two requests to the server to work on particular file: first when you download a file and a second when you send an updated file back to the server. So, it saves server resources and provides better performance.

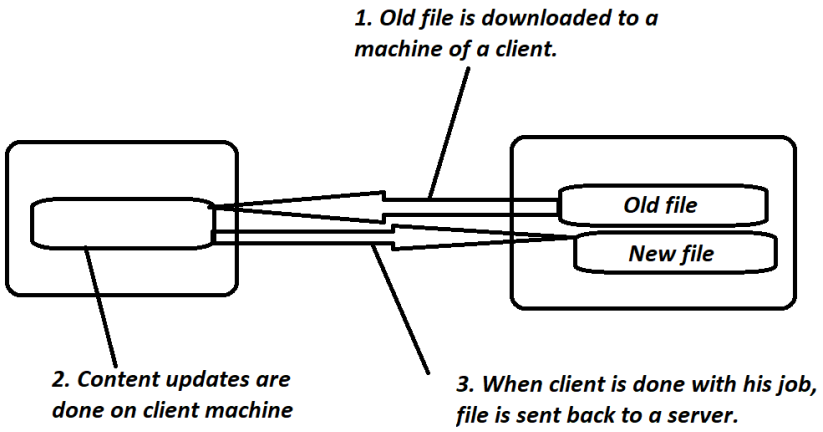

Figure 2. Upload/Download model.

Another useful benefit of this model is that you can use a file even if you cannot access the server. A disadvantage of downloading file to the local system and then sending it back to the server is that concurrent modification of a file by di erent clients can cause problems.

The Remote-Access model provides a possibility for the file server to order all operations and allow concurrent modifications to the files. A disadvantage of this model is that client could use a files if it can connect to the file server. If there will be no connection to the server then client will be not able to access the files.

2.2 Cluster-based architecture

DFS with cluster-based architecture do not store data on a single server. There are multiple servers in such a system. One of them is the master server. The purpose of this server is to store the metadata of data. All other servers are chunk servers. With having more than one server, Chunk could handle multiple clients at the same time. Using such an architecture, very large data can be processed. Example of cluster-based structure could be a Google File System [10] (see 3 Fig.).

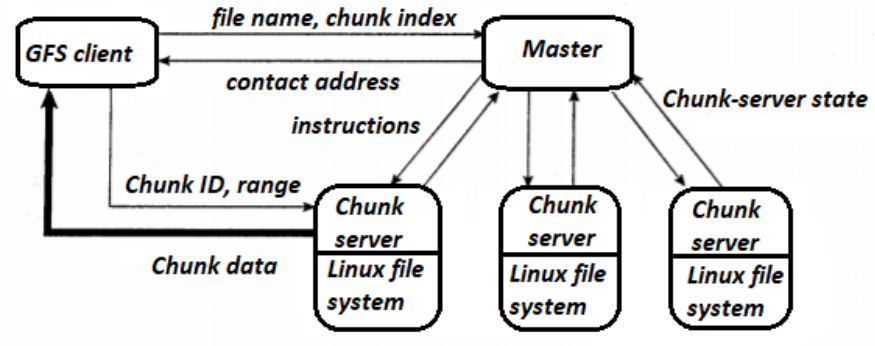

Figure 3. Cluster-based architecture.

2.3 Symmetric architecture

Fully distributed file systems also known as decentralized do not have servers. It means that such a systems are made up from a lot of nodes which shares data between each other. Majority of systems use Distributed Hash Table (DHT) approach for distributing data between nodes. An

example of such an architecture could be Ivy Distributed File System (see 4 Fig.). It consists of three layers: the first layer is distributed hash table used to generate hash of files, the middle layer which is fully distributed block-oriented storage layer and third layer which spans a whole file system [10].

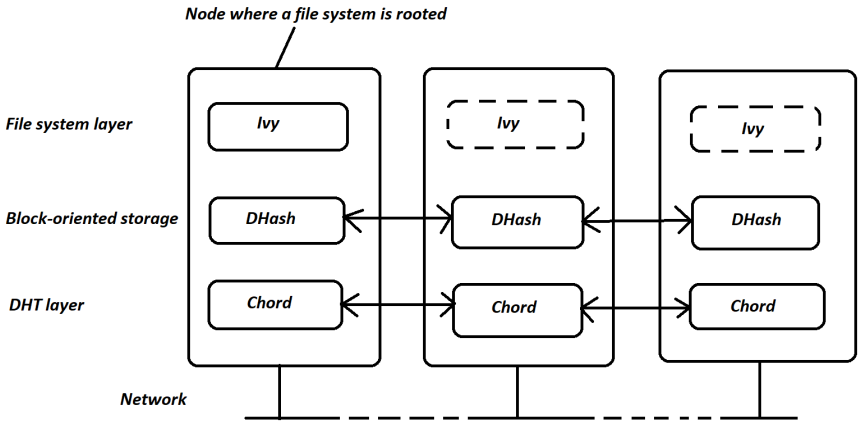

Figure 4. Symmetric architecture.

3 Introduction to Interplanetary File System

IPFS is known as InterPlanetary File System. The development of this system was not started from a base. IPFS combines ideas from previous p2p systems like BitTorrent, Git and Kademlia DHT. Developers wanted this system to become a change of systems such as HTTP. IPFS and HTTP are compared below(see 5 Fig.) [17]. All members of IPFS are known as nodes. IPFS file sharing takes place directly between nodes. IPFS is a p2p system without servers. The main di erence between these two systems is that IPFS do not have server, and HTTP has it [1].

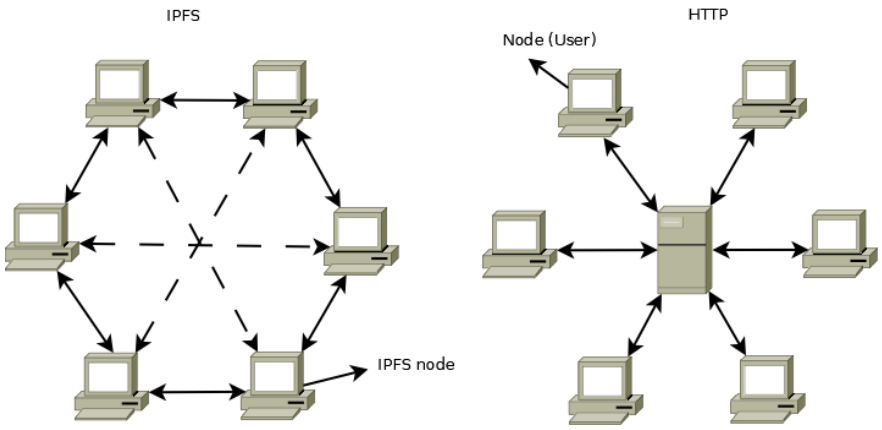

Figure 5. Di erence between IPFS and HTTP.

3.1 Developers of IPFS

Juan Benet and company Protocols Labs are founders of IPFS system. The development of system was started in 2014 and being developed up to now [2].

4 Definitions

This section will help you to get familiar with the definitions used in this work.

4.1 Data Addressing

HTTP is a mechanism used for data addressing with URLs. In IPFS system data addressing is carried out using contents hash.

4.1.1 HTTP data addressing

Hostname, port, path and the filename are used to address data in HTTP [3]. For example, if the location of a certain file is changed, and no other page was created for that link, then that link becomes unreachable.

4.1.2 IPFS data addressing

Files in IPFS can be accessed by using the content hash. Content hash is used to build the link to files. If we add a file to IPFS, the certain hash will be assigned to that file. IPFS provides an ability to change files content (Git property) [1]. For example, if we add a file X.txt with a content of "I like IPFS" and getting a hash of it, then change a content to "I like IPFS very much" and adding it one more time to IPFS then a new hash will be created. The both versions of file including old and new will be still available only with separate hashes. That will be demonstrated in **7.2** section.

Example 4.1.1. *IPFS hash example*

*https://ipfs.io/ipfs/**QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX***

4.2 Multiformats

Multiformat is a set of protocols/formats used in IPFS. They role is to try to extend existing formats by adding self-describing components. Two formats that are part of the IPFS structure are **Multihash** and **Multiaddr** [8].

4.2.1 Multihash

Multihash is a multiformat or protocol/format maintained and developed by Protocol Labs company. This multihash hash function is used for IPFS Hashes encoding. Every hash-key consists of it and two additional values: the function code of the hash function used to create this hash and the length of hash. IPFS nodes use sha2-256 hash function. After generating multihash it will be encoded by Base58 [9].

Example 4.2.1. *IPFS hash example*

sha2-256	***size***	***sha2-256("hello ipfs")***
0x12	*0x20*	*String multiHash*

*multiHash = "**4779eac1a794c2777e06cd6300e8faa3801b0591a291e581f2d6a20c19fd38da**"*
After concatenating that three columns it will be:
*concated = "**12204779eac1a794c2777e06cd6300e8faa3801b0591a291e581f2d6a20c19fd38da**"*
After encoding it to Base58 it will be:
*finalHash = "**QmT9fUoLK3juU5drvgSomPxNDiJZuzmH4sYpYhVyDNFefB**"*

4.2.2 Multiaddr

Multiaddr is way to express network addresses to a di erent combinations and is mostly used in nodes addressing. Similarly as in multihash every function has its own function code, in Multiaddr every protocol represents a function code [7].

Example 4.2.2. *Typical form:*
/ip4/127.0.0.1/udp/1234
Multiaddr form:
***0x4 0x7f 0x0 0x0 0x1 0x91 0x2 0x4 0xd2**.*

5 Mostly used IPFS commands in Linux terminal

As we already know IPFS allows us to share and get (download) desirable content. Linux terminal provides an ability to execute IPFS commands quickly and comfortably. Before executing any IPFS statement it is necessary to initialize IPFS configuration and start a daemon processes which are needed to start working with IPFS.

Example 5.0.1. *Initialization of IPFS node and starting of daemon process.*
$ ipfs init && ipfs daemon

To add or get data from IPFS system user has to perform **add** and **get** commands respectively. Lets say we have a text file **test.txt** with a content of "hello ipfs".

Example 5.0.2. *Example of adding data to IPFS.*

Listing 1. Adding data to IPFS.

```
1 ipfs add test.txt
2 added QmSoASxb8aNVGk3pNWpZvXEZTQKxjGeu9bvpYHuo5bP1VJ test.txt
```

As we see hash of the file is generated.

Example 5.0.3. *Example of getting data from IPFS.*

Listing 2. Getting data from IPFS.

```
1 ipfs get QmSoASxb8aNVGk3pNWpZvXEZTQKxjGeu9bvpYHuo5bP1VJ
2 ls
3 QmSoASxb8aNVGk3pNWpZvXEZTQKxjGeu9bvpYHuo5bP1VJ
```

After get command is successfully executed the requested file will appear in a current directory as we see and also it could be accessed from a browser:
https://ipfs.io/ipfs/QmSoASxb8aNVGk3pNWpZvXEZTQKxjGeu9bvpYHuo5bP1VJ

6 IPFS-Architecture

6.1 Identities

Identities sub-protocol is responsible for the node identity and verification of IPFS nodes. When the IPFS node is initialized, then a 2048-bit RSA key pair(public and private keys) is created. Public key hash is generated using a multihash function and then this hash is assigned to a NodeID [18].

6.1.1 Communication between Nodes

When a first node tries to establish relationship with second node, both nodes exchange NodeID and Public key. First node generates hash of the second node public key and if that hash is not the same as NodeID of a second node than connection between these two nodes will be interrupted [1].

6.2 libp2p - PeerToPeer library

This p2p library holds a content used for **network** connectivity between nodes, **routing**(data lookup) in the network and data **exchange** between nodes [14].

6.2.1 Network

This layer is responsible for connection to other nodes in the IPFS file system. Goal of this libp2p part is to get the maximum IPFS node availability level over the whole IPFS network [1].

6.2.2 Routing

Routing provides an opportunity to get a specific network addresses of other IPFS nodes that can provide desirable data.

The routing module of node gets a hash-key and get respond with the list of one or more node info objects which contain a NodeID and multiaddr addresses of that node. This data is used for contacting the other IPFS nodes [1].

Distributed Hash Table

Distributed Sloppy Hash Table (DSHT) is used to hold hash key value of NodeIDs pairs. It is distributed on all active IPFS nodes. It means that every node will be responsible for a certain part of all blocks added to IPFS. If IPFS node has no more connection to IPFS, then the responsibility of his part of the DHT is given to other node [1].

Files are stored inside IPFS objects and these objects can store up to 256 kilobytes worth of data.

If file is bigger than 256KB like a video or image then that file is split up into multiple IPFS objects that are all 256 kilobytes size. After that the system will create an empty IPFS object that links to all the other pieces of the file. If more technically for every object new entry with content of a block hash with reference to a NodeID is added to DHT.

Example 6.2.1. *Example of adding a file bigger than one IPFS object size (256kb).*

Listing 3. Requesting IPFS objects of file > 256kb.

```
1 ipfs get object QmdJTmCxwcpoGbEVfT6b9j4RZJWNcF2GQG1Ajf9XB6XtVP
```

And in a output we get:

Listing 4. Created IPFS objects of file > 256kb.

```
1 "Links":[
2 {"Name":"","Hash":"QmTi2QvnQgwdnQEide1GtZ4WrmJgrxerzTp4MNei9UJC2w","Size
      ":262158},
3 {"Name":"","Hash":"QmZpwJYecK6xYw8P9YsJyRo5a7LJFEckHEXxSqP178vQyJ","Size
      ":262158},
4 {"Name":"","Hash":"QmdQUyndMnU9sXAvPbNNZbphWBfGWoLB7wHg4NGm7VpWkf","Size
      ":255406}],
5 "Data":"\u0008\u0002\u0018\ufffd\ufffd/ \ufffd\ufffd\u0010 \ufffd\ufffd\
      u0010 \ufffd\ufffd\u000f"}
```

For example, text file with the content of "hello ipfs" can be stored in a single IPFS object because its size do not exceed 256KB. A new entry with content of block hash with reference to File is added to DHT.

Example 6.2.2. *Example of adding a file which size is less than one IPFS object size (256kb). We will use the same test.txt file used for examples in previous section.*

Listing 5. Requesting IPFS objects of file < 256kb.

```
1 ipfs get object QmSoASxb8aNVGk3pNWpZvXEZTQKxjGeu9bvpYHuo5bP1VJ
```

And in a output we get:

Listing 6. Created IPFS objects of file < 256kb.

```
1 {"Links":[],
2 "Data":"\u0008\u0002\u0012\u000bhello ipfs\n\u0018\u000b"}
```

6.2.3 Exchange

IPFS distributed file system can use a lot of exchange protocols. The mostly used protcol in IPFS is BitSwap. BitSwap is used to perform data transfer between the IPFS nodes. BitSwap functionality consists of such a steps:

1. Requester IPFS node asks for a certain part of file to download.
2. That file is added to want list of that first IPFS node.
3. BitSwap module sends that want list to other nodes that have some parts of that file according to DHT.
4. That nodes put their having parts of data to the have list.
5. If one of that nodes has that file that the first node wanted to download then that part of data is sent to requester node.
6. At the moment when desired part of data is sent all other nodes will stop searching for data.
7. Want list and have list are eliminated.

7 Related work

As mentioned in the beginning, the purpose of this work is to focus on IPFS performance. We will do a simple download test case which is written in Bash language. We have five Ubuntu 16.04 virtual machines with IPFS installed. Goal is to test IPFS performance by choosing a di erent number of requests and delays between them. For defining performance test results, we will use average request execution time variables. We will perform three tests:

- 5 requests to the file with delay which is equal to 0 seconds. It means that we will send 5 requests to the file without interrupts.
- 5 requests to the file with delay which is equal to 5 seconds. It means that we will send 5 requests to the file every five second.
- 5 requests to the file with delay which is equal to 10 seconds. It means that we will send 5 requests to the file every ten seconds.

After this test we will check di erence between all three download speed averages.

7.1 Download Experiment

Listing 7. Download test with delay which is equal to 0 seconds.

```
1 Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
2 208.44 KB / 208.44 KB  100.00% 0s
3
4 Download time of IPFS_NODE1: 1.313395512 seconds
5 Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
6 0 B / 208.44 KB    0.00%
7 208.44 KB / 208.44 KB  100.00% 0s
8 Download time of IPFS_NODE2: 1.449446200 seconds
9 Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
10 208.44 KB / 208.44 KB  100.00% 0s
11
12 Download time of IPFS_NODE3: 1.277026253 seconds
13 Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
14 208.44 KB / 208.44 KB  100.00% 0s
15
16 Download time of IPFS_NODE4: 3.003086492 seconds
17 Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
18 0 B / 208.44 KB    0.00%
19 208.44 KB / 208.44 KB  100.00% 0s
20 Download time of IPFS_NODE5: 1.050746487 seconds
21
22 Average download speed of all download processes in kb/s:
23 146.70291
```

Result of this test is that average download speed of all five requests is **146.70291** kb/s.

Listing 8. Download test with delay which is equal to 5 seconds.

```
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
0 B / 208.44 KB     0.00%
208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE1: .931403434 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX

208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE2: .834524438 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
0 B / 208.44 KB     0.00%
208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE3: 1.158473146 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
0 B / 208.44 KB     0.00%
208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE4: 5.835610128 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
208.44 KB / 208.44 KB  100.00% 0s

Download time of IPFS_NODE5: 2.079382666 seconds

Average download speed of all download processes in kb/s:
157.88973
```

In this test we can see that average download speed is equal to **157.88973** kb/s.

Listing 9. Downlaod test with delay which is equal to 10 seconds.

```
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
208.44 KB / 208.44 KB  100.00% 0s

Download time of IPFS_NODE1: 1.318359470 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
0 B / 208.44 KB     0.00%
208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE2: .902398822 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX

208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE3: 1.534072257 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX
208.44 KB / 208.44 KB  100.00% 0s

Download time of IPFS_NODE4: 1.293983729 seconds
Saving file(s) to QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX

208.44 KB / 208.44 KB  100.00% 0s
Download time of IPFS_NODE5: 1.671444723 seconds

Average download speed of all download processes in kb/s:
162.15079
```

As seen in the example the average download speed is **162.15079** kb/s.

7.2 Results of Download Experiment

As we can see by increasing the delay between downloads, the average download speed is increasing. This means that it is harder for the system to distribute the files continuously requesting to download a file. Increasing delay file download speed becomes faster. It is also necessary to take into account the speed of the internet, because results can be completely di erent when internet speed changes.

7.3 Upload Experiment

To test performance of a system, it is logical to test not only speed of download process but also how fast system is able to upload data. Now we will measure it by average upload time. We will perform such a test case:

- 5 requests to upload a file with delay which is equal to 0 seconds. Five requests will be sent without interrupts.
- 5 requests to upload a file with delay which is equal to 5 seconds. Five requests will be sent every 5 seconds.
- 5 requests to upload a file with delay which is equal to 10 seconds. Five requests will be sent every 10 seconds.

Listing 10. Upload test without delay.

```
1 Please enter a delay of each upload process
2 0
3 Please enter a number of upload processes
4 5
5
6 added QmXr4EEva2mz3U1uJ5DfhonRdNcghkCsLyr9sdSgmwVUpj test.txt
7 added QmSmvKc3hu3riNzneobQuQ3ZkaAw7oRKuJz5KxyQEqQkn2
8
9 added QmSw4k8zvbxrFWWT5B9MHLuQ2xPunbdJBxAtWWsXt5e5og test.txt
10 added QmNWmLXn2cGoPetjVP8y4nrf9S4HnQm15Li8igaGRXFoP1
11
12 added QmbScFm2rSvbReKUornBJVMgakaqJYm4ZD2mTTamyXNZrj test.txt
13 added QmVCBfdAdybJcPTuQFqWGwqCBLXwtVLeXWPTwZPM2HK7Rb
14
15 added QmV2UFkxh4S5YMQysXExAtZqCyYwyvnzo1rP8FdpKyuQKt test.txt
16 added QmXCaQnkeCDy4mf9DVtzU1NH5f2rMfNquYGR3YaAbymrUn
17
18 added QmPQttZb6F3a1i8rLv3EJ8AK36uMQtRximpR9mqbN5WUMt test.txt
19 added QmPVvZGc4RAeroHVKYvjPrBW821w1BMcXz7FuwLCqyQHj9
20
21 Sum of all upload processes: 6.558775291
22
23 Average upload speed of all processes: 1.31175
```

As we see the average of all upload processes is **1.31175** seconds.

Listing 11. Upload test with delay which is equal to 5 seconds.

```
1 Please enter a delay of each upload process
2 5
3 Please enter a number of upload processes
4 5
5
6 added QmXr4EEva2mz3U1uJ5DfhonRdNcghkCsLyr9sdSgmwVUpj test.txt
7 added QmSmvKc3hu3riNzneobQuQ3ZkaAw7oRKuJz5KxyQEqQkn2
8
9 added QmSw4k8zvbxrFWWT5B9MHLuQ2xPunbdJBxAtWWsXt5e5og test.txt
10 added QmNWmLXn2cGoPetjVP8y4nrf9S4HnQm15Li8igaGRXFoP1
11
12 added QmbScFm2rSvbReKUornBJVMgakaqJYm4ZD2mTTamyXNZrj test.txt
13 added QmVCBfdAdybJcPTuQFqWGwqCBLXwtVLeXWPTwZPM2HK7Rb
14
15 added QmV2UFkxh4S5YMQysXExAtZqCyYwyvnzo1rP8FdpKyuQKt test.txt
16 added QmXCaQnkeCDy4mf9DVtzU1NH5f2rMfNquYGR3YaAbymrUn
17
18 added QmPQttZb6F3a1i8rLv3EJ8AK36uMQtRximpR9mqbN5WUMt test.txt
19 added QmPVvZGc4RAeroHVKYvjPrBW821w1BMcXz7FuwLCqyQHj9
20
21 Sum of all upload processes: 5.534382174
22
23 Average upload speed of all processes: 1.10687
```

In this case average of upload processes is **1.10687** seconds.

Listing 12. Upload test with delay which is equal to 10 seconds.

```
1 Please enter a delay of each upload process
2 10
3 Please enter a number of upload processes
4 5
5
6 added QmXr4EEva2mz3U1uJ5DfhonRdNcghkCsLyr9sdSgmwVUpj test.txt
7 added QmSmvKc3hu3riNzneobQuQ3ZkaAw7oRKuJz5KxyQEqQkn2
8
9 added QmSw4k8zvbxrFWWT5B9MHLuQ2xPunbdJBxAtWWsXt5e5og test.txt
10 added QmNWmLXn2cGoPetjVP8y4nrf9S4HnQm15Li8igaGRXFoP1
11
12 added QmbScFm2rSvbReKUornBJVMgakaqJYm4ZD2mTTamyXNZrj test.txt
13 added QmVCBfdAdybJcPTuQFqWGwqCBLXwtVLeXWPTwZPM2HK7Rb
14
15 added QmV2UFkxh4S5YMQysXExAtZqCyYwyvnzo1rP8FdpKyuQKt test.txt
16 added QmXCaQnkeCDy4mf9DVtzU1NH5f2rMfNquYGR3YaAbymrUn
17
18 added QmPQttZb6F3a1i8rLv3EJ8AK36uMQtRximpR9mqbN5WUMt test.txt
19 added QmPVvZGc4RAeroHVKYvjPrBW821w1BMcXz7FuwLCqyQHj9
20
21 Sum of all upload processes: 4.934000186
22
23 Average upload speed of all processes: 0.98680
```

Average of upload processes in this test case is **0.98680** seconds.

7.4 Results of Upload Experiment

First thing that should be noticed in this experiment is that hash changes by every upload. We use an algorithm which changes a content of test.txt file to be able to mark it as an updated content and generate a new hash in IPFS system. As we see the average of upload times get shorter by reducing delay time. It means that it is more di cult for the system to upload a content without interrupt (with delay which is equal to 0 seconds). As in download test here is also necessary to take into account the speed of the internet, because results can be completely di erent when it changes.

Conclusions and Recommendations

InterPlanetary File System is a near future technology which makes internet fully distributed and decentralized. InterPlanetary File System can be considered as a system having a consistent architecture which makes a content as available as possible. Experiments have shown that IPFS performance depends on load - the more queries to the file, the slower the content gets to the physical machine.

References

[1] Juan Benet. IPFS-content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014.

[2] Juan Benet. PROTOCOL LABS: CREATING NEW NETWORKS, 2017.

[3] Tim Berners-Lee. HTTP Addressing.

[4] Benjamin Depardon, Gaël Le Mahec, and Cyril Séguin. Analysis of six distributed file systems. 2013.

[5] Claudia Eriksen. Comparison of OpenAFS, NFS and Samba. Master's thesis, 2005.

[6] Glenn Fleishman. *Take Control of Sharing Files in Snow Leopard*. Take Control Books, 2009.

[7] Protocol Labs Inc. Multiaddr, 2016.

[8] Protocol Labs Inc. Multiformats, 2016.

[9] Protocol Labs Inc. Multihash, 2016.

[10] Eliezer Levy and Abraham Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys (CSUR)*, 22(4):321–374, 1990.

[11] J.C.R. Licklider. In Memoriam: JCR. Licklider. *Technology*, 1968.

[12] Sudheer R. Mantena. Transparency in distributed systems. *CSE*, 6306:13.

[13] Kovendhan Ponnavaikko and D. Janakiram. The Edge Node File System: A distributed file system for high performance computing. *Scalable Computing: Practice and Experience*, 10(1), 2009.

[14] Mark Pors. Understanding the IPFS White Paper part 2, 2017.

[15] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: principles and paradigms*. Prentice-Hall, 2007.

[16] Miss J Vini, Rachel Nallathamby, and CR Rene Robin. A novel approach for replica synchronization in hadoop distributed file systems. *Procedia Computer Science*, 50:590–591, 2015.

[17] Peter Vowell. What is the InterPlanetary File System?, 2016.

[18] Matt Zumwalt. How Does Node Identity Work?, 2017.