applied sciences

MDPI

Article

A Distributed File-Based Storage System for Improving High Availability of Space Weather Data

Yoga Andrian [1,2], Hyunwoo Kim [1,*] and Hongtaek Ju [1,*]

[1] Department of Computer Engineering, Keimyung University, Daegu, South Korea
[2] Space Science Center, National Institute of Aeronautics and Space (LAPAN), Bandung, Indonesia
* Correspondences: juht@kmu.ac.kr

Abstract: Currently, a centralized storage model that implemennted for supporting space weather forecast system has some problems that impact to researchers as the primary users. Single point of failure and also the delay in data updating on the server is a significant issue when researchers need the latest data, but the server is unable to provide it. To overcome these problems, we propose a new system that utilizes a decentralized model for storing data leveraging IPFS filesystem. Our proposed method focuses on the automated background process, and its scheme will increase the data availability and throughput by spreading it into nodes through a peer-to-peer connection. Moreover, we also include system monitoring for real-time data flow from each node and information of node status that combines active and passive approaches. For system evaluation, the experiment is performed to determine the performance of the proposed system compared to the existing system by calculating mean replication time and the mean throughput of a node. As expected, performance evaluations show that our proposed scheme has faster file replication time and supports high-throughput.

Keywords: distributed file; peer-to-peer network; automated process; decentralized storage; data management

1. Introduction

With the increase of numerous research efforts related to the space environment, the advancements of space technology greatly influence human life in the current era. One of the most important examples is the utilization of satellites for communication and navigation. Currently, space research has been conducted by many countries showing high seriousness by establishing a national space agency that intensively supports government programs in space science and technology development. Indonesia, represented by National Institute of Aeronautics and Space [1], is one of the countries in the Association of Southeast Asian Nations (ASEAN) that is actively researching aeronautics developments and space weather activities. In space research, LAPAN has an information system service that provides data, information, and forecasts related to space weather called Space Weather Information and Forecast Services (SWIFtS). SWIFtS is one of the space weather services approved as an ISIS member for the ASEAN region since 2016. The SWIFtS website contents are daily analysis results by forecasters that include solar activity, geomagnetic activity, and ionospheric activity as well as one-day or three-day forecasts at the weekend. Other essential information is also provided to the space weather user community, such as the current conditions on high-frequency radio communication propagation, the position error percentage for navigation, and the number of proton energies from the sun that impact satellites and high-frequency predictions [2].

SWIFtS researchers and forecasters conduct daily data analysis and processing to provide accurate information on the SWIFtS website for the user. For this reason, the data availability from space weather observation instruments is continuously updated to maintain information. At the

background process, SWIFtS is supported by the data storage system that collects the data from various instruments installed on observatory stations spread throughout areas in Indonesia. The daily information provided by SWIFtS requires the data storage system to be continuously running well. The existing system implements the client-server hierarchy based on the centralized model. However, the process that has been running did not meet our expectations. The delay in data updating on the server is a significant issue and causes problems when researchers need the latest data but the server is unable to provide it. This is due to the centralized model that have implemented in the current system. Each time a personal computer (PC) collected data from the instrument, it sent the data to a server located in the observatory stations.

There are eight observatory stations (hereafter referred to as sites) spread in different islands and one central data center (Figure 1). Researchers in Bandung, where the data center is located, need actual data for data processing and making forecasts to update the SWIFtS website. Problems arise when servers in Bandung must provide the latest data while the site server is down. The researchers have trouble conducting their research, and the renewal of information on the SWIFtS website is hampered.

Figure 1. Map of LAPAN observatories in Indonesia.

Centralized data management is a conventional method to store data that relies on one single server. A major drawback of this approach is the data availability issue; access to data is limited only by server lifetime and the connection between the user and the server. Moreover, the centralized storage system can represent a high risk of data loss without an optimal backup system. Further investigation reveals that, in addition to being a single point of failure, other issues that cause data updating delay in centralized storage include heavy traffic volumes and link bottlenecks. Moreover, the instruments that have generated the latest data cannot update the server dataset, and the data growth increases daily requiring adequate storage capacity.

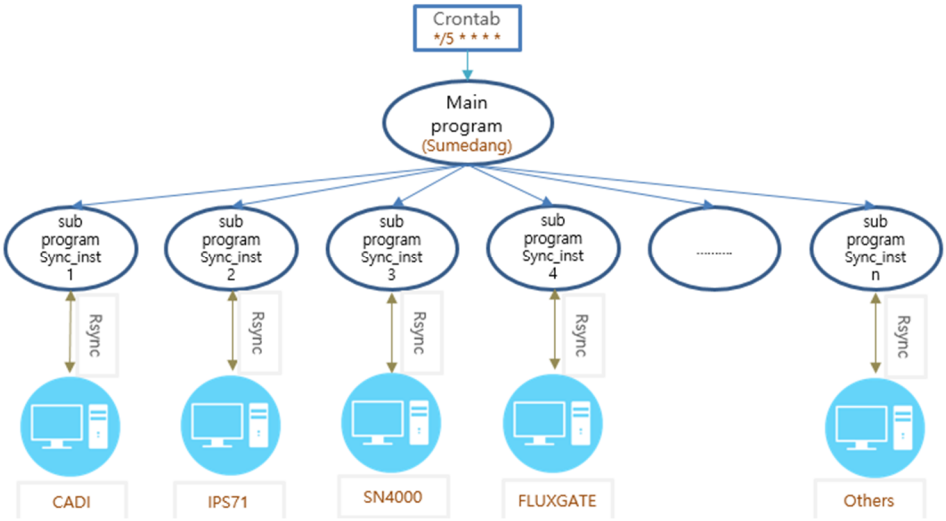
Hence, a new solution that can address these problems is needed. To improve data availability, we should move to a decentralized storage system. This can benefit by leveraging available resources from other hosts, taking advantage of high data throughput, while also mitigating single point of failures. The decentralized storage is a method of storing data by encrypting and distributing the data across a decentralized network. It does not rely on a central service provider for data storage. Decentralized storage can eliminate and solve the problem of data failures [3] and outages in the traditional storage system by increasing security, privacy, and control of the data [4-6]. Moreover, decentralized storage also can increase the percentage of data availability by spreading files to be stored on the hosts that are connected peer-to-peer. Consequently, this method, where each host can exchange files directly as well as have roles as a client and server, would reduce user dependency on a single server. In addition, encryption by a hash function offers an advantage of compressing files and protecting from data corruption. Thus, data generated by the instruments can be kept on each data collection server continuously without concern about single point failures while also providing improved data security [7].

The Interplanetary File System (IPFS) is a peer-to-peer decentralized system protocol for distributing files throughout connected computing devices in the same system of files. The IPFS provides a high data throughput with a content-addressed block storage model, requires no central server, and distributes and stores the data in spread locations [8-10]. In this study, a new system is proposed for data management using a decentralized storage model leveraging the IPFS distributed system for storing and sharing the space weather observation data across the peer-to-peer network. Moreover, a technique for monitoring the system through active and passive approaches is developed. The passive approach is intended to gain the flow of data between nodes, and the active approach determines the status of each node (e.g., whether online or not). Afterward, system performance is evaluated and analyzed by comparing the performance of the proposed system with that of the existing one. The tested parameters are the mean time of replication of files and the mean throughput from a node.

The remainder of this thesis is organized as follows. Section 2 discusses the background of this study, and section 3 provides the architecture of the proposed system. Section 4 describes the detailed implementation of the system. Section 5 deals with the experiments and results of this work. Finally, the conclusion and future direction are presented in section 6.

2. **Background**

The rapid growth of data provides great benefits for its users; thus, data is becoming a valuable asset and must, therefore, be properly maintained [11, 12]. Furthermore, a reliable storage system is also needed to provide high availability of data continuously that meets user needs. Space weather researchers in LAPAN need data from space weather observations as input to their research. When the necessary data is not available, production of the latest information is hampered. We indicate that the centralized method of the existing system has several drawbacks, such as low data availability, low throughput, and increased time for data updates. Reliance on one server can result in a single point of failure when the destination server is down. Currently, the main server in Bandung is the central destination for storing data. A large number of users accessing the Bandung server can create a high network load on the server



.Figure 2. The synchronization process between site server and instrument PC at Sumedang site.

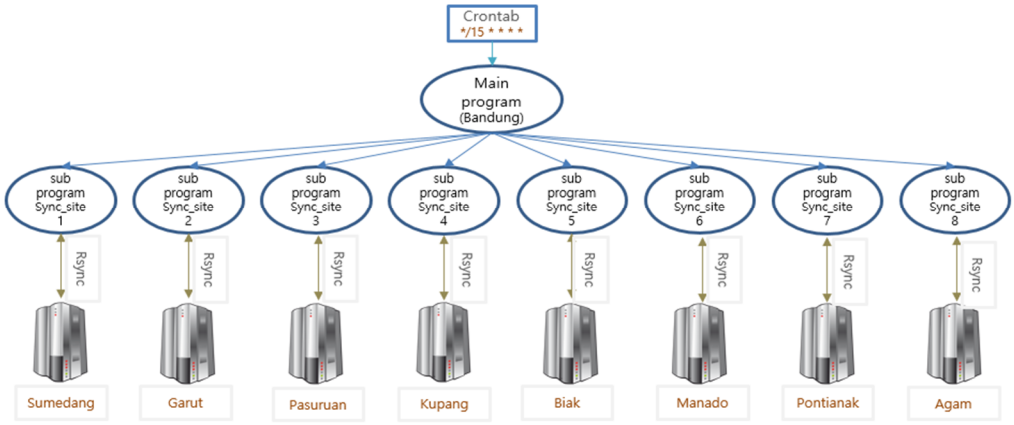


Figure 3. The synchronization process between central and site server.

Furthermore, the process of synchronizing data between the Bandung server and all the site servers at the same time also affects network performance on the server. If we elaborate on those factors, the main problems associated with data updates arise from synchronization at the same time from the central server to all site servers or between site servers and PC instruments. The current system employs two crontabs (cron tables), which are configuration files that specify shell commands for running a script program on a schedule. First, a script on the site server runs the main program every 5 min for data synchronization between the site server and all instrument PCs (Figure 2). Second, a crontab job on the central server runs every 15 min. It synchronizes the data between the central server and all site servers (Figure 3). All synchronization processes run at one time to synchronize all data through default SSH port utilizing the Rsync tool. Thus, this process creates a queue of jobs that have to be executed while the various data from each site and instrument should be stored safely in real-time and without damage. This lowers throughput in sending data because of the scripts running at the same time on one port. Low throughput affects the duration of data synchronization due to the high network load; thus, synchronization takes longer. A delay in updating files also occurs when data with small size cannot be sent immediately because of the need to wait for the larger data in the queue to finish.

Of course, these situations emerge as a problem when all concurrent access is directed to one port. The queue builds and causes delay for data updates due to the high network load. In our proposed scheme, in addition to spreading the processes, the scheduler was replaced with a real-time directory watcher on each site server and instrument PC. Data monitoring has been implemented in the current system; however, it has not been arranged neatly and there is no web interface that makes it easy for administrators to observe the running process. The present work created a real-time web interface to display details of file metadata for each node that is distributed to others and the status of nodes whether they are connected to IPFS or not.

2.1. Related Works

This section reviews related works that leverage decentralized network approaches to store the data; however, this approach needs the user as an actor to upload or download the data manually. Most of the papers explained the utilization of the IPFS as their storage system combined with blockchain. However, they did not disclose detailed information on how the IPFS works.

Sia is a simple decentralized storage system proposed by Vorick et al. [13]. They need clients as users to upload and download files in the Sia network; there is no automated process involved with getting or putting data. Likewise, there is Storj, a peer-to-peer cloud storage network [14]. Wilkinson, assisted by some contributors, has built Storj by proposing a Proof-of-Storage. However, this system also needs actors to upload and download files manually from the system. Similar to Sia and Storj, Maidsafe comes with other demands for storing data in the decentralized and secure network [15] and needs a manual process to store and retrieve the data. Sia, Storj and Maidsafe rely on the blockchain network and have a more commercial focus, needing cryptocurrency to use their service.

In 2018, Nygaard [16] leveraged the IPFS-Cluster to limit the replication of data only for peer members. His proposed system also needs the client as actors for storing data manually. However, our proposed system does not need cryptocurrency or tokens because there is no commercial orientation in this system. Our focus was only on file exchange without coin/token utilization. Moreover, this system also requires an automatic process to store and retrieve the data. This is because various data are produced by the instruments continuously, such as every 5 min, 15 min, hourly, and daily. It would be impossible for a human to conduct this as a manual process.

The building of a scaled-out Network Attached Storage (NAS) and the IPFS to store an object spread through the Fog/Edge infrastructure has been proposed by Confais, et al. [9]. This system implemented a Proof-of-Concept using RozoFS with modifications in the IPFS source code. However, this system did not provide a detailed explanation of how the IPFS works and what protocol the IPFS uses. Another work proposed decentralization for big data by Brisbane [17]. He leverages the IPFS by changing the Hadoop Distributed File System (HDFS) as the file system. He did not give a detailed explanation of how the IPFS stores and retrieves the data. In this thesis, a detailed explanation is provided of how the IPFS works as a distributed file system using a P2P network.

Indeed, the IPFS provides a web user interface on each node connected to the network. One of the features that displayed is all global peerIDs that are connected to the IPFS. However, our system only needs status information from all nodes in the cluster and data flow from each node added to the IPFS. In addition, we need a centralized monitoring system that can determine the current condition of all peers in our cluster that the IPFS does not have today.

3. **System Design**

This section analyses the design of the proposed system such as system architecture, and network diagram. We also explain one sample process of the sequence diagram and show flow diagrams of our monitoring approaches.

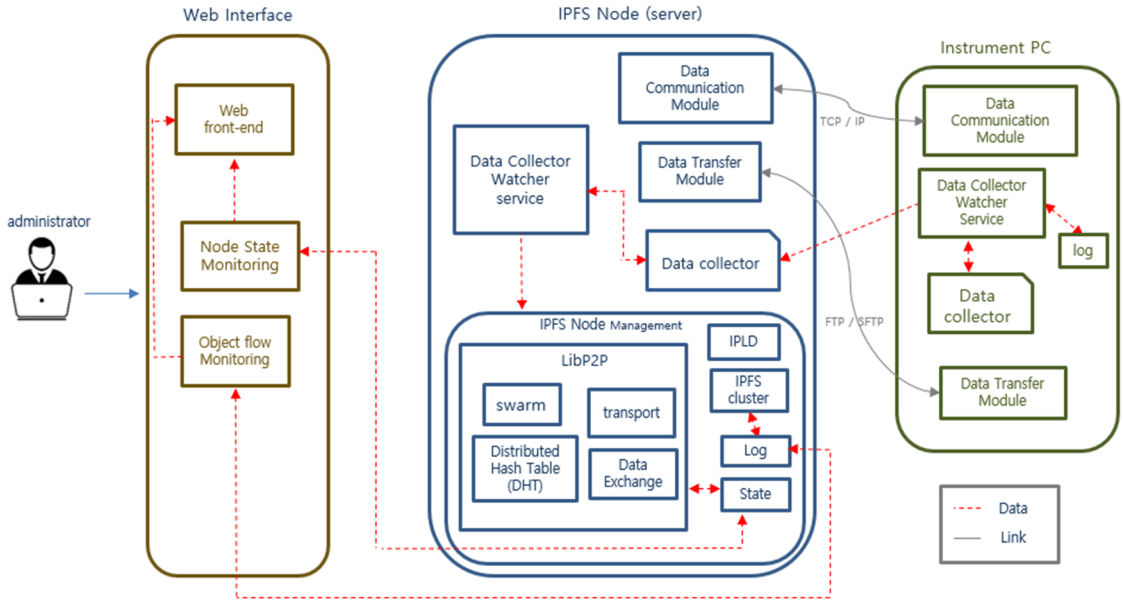
3.1. System Architecture

The IPFS allows any node that has a connection to the network to be able to retrieve and store data for themselves by "pinning" content actively [17]. This thesis utilized the IPFS as our core system to store data and distribute it to other nodes. In the IPFS, initializing the repository generates a key pair (public and private key) and forms a local folder that hosts the IPFS configuration and stores repository objects from a node. The identity of each node (NodeID) is made by using a hash of the node's public key.

The proposed system is making the central and site server as a node connected to the IPFS network in a swarm. Peers are restricted to a private IPFS network that has been prepared for the process of distributing files between peers automatically. The IPFS can be seen as an object storage service on top of the Bit-Torrent protocol because the fundamental process of exchanging data between peers is similar to it [8]. When a file is published on the IPFS, peers can access that content to keep in the local copy. Data becomes publicly available for at least 24 h; if there is no interaction (pinning) from other peers, data will be thrown to the garbage collector without recycling. The garbage collector tool is used to clear unused data in the network. In our proposed system architecture, it is necessary to handle the sharing of files by automatic pinning since the instruments generate various and continuous data.

The data are formed daily, hourly, every 15-min, and every 5-min depending on its type. Every site also has various instruments, but several of them installed are similar to those at other locations such as a Canadian Advanced Digital Ionosonde (CADI), which is a special radar for studying ionized parts in the Ionosphere layers. It is installed in several locations and is always producing data per 15-min intervals. To keep data available, it must be distributed between peers as soon as the data

is formed. For this reason, another scheme is used to do automatic pinning for every file added to the



IPFS.

Figure 4. The proposed system architecture.

IPFS-Cluster is a tool used to coordinate between IPFS daemons running on different hosts [18]. If they are interconnected in one cluster, then every object added by a peer in IPFS will be pinned by all members automatically. So, if one peer disconnected from the network or delete that object, it still provided by others. However, the peer receives only the piece of them, and once peer distributes an object, it's no more duplication in the network and still be an object with the same hash (deduplication). Thus, this scenario may not significantly increase the storage space of the nodes since each file with identical content will not be stored. Moreover, the beneficial will comes up since employing available resources on nodes.

The instrument PCs are included as parts of this system because they are the initial contributors that collect data from the instruments. They have two types of operating systems installed, Windows and UNIX. Figure 4 shows the architecture of the proposed system. The data communication module between server and PC is using TCP/IP. For the data transfer module, this system separates it based on the PC's operating system. FTP is used for data transfer to the Windows OS, whereas SFTP is used for the PC which has the Unix OS installed. Each of them has a data collector as a place where data is stored. The data collector watcher has been used on the instrument PC for sending the latest file automatically to the data collector on the site server (IPFS node) and then generates a log file containing its metadata. This watcher is built with the watchdog library from python.

On the IPFS node, a data collector watcher service, that was essential to keep the latest data available, was also employed. This service was built using the pyInotify library from python and a client library of the IPFS API. A distinct root directory based on the instrument's name while on instrument PC was set as the root directory which stores the raw data. The program code used a threading technique to be a separate process for monitoring on each root directory. When the new file is produced and sent by the instrument, it is collected to a folder on the server which named according to the instrument type. The program adds data to the IPFS whenever a there is a change in the folder and generates a log containing metadata information such as filename, timestamp, the instrument's name, block hash, file size, and total size. In Figure 4, the server runs the IPFS binary to manage its resources as an IPFS node. LibP2P is the major part of our system that requires some mandatory tasks such as allowing for data and communication transport, creation of a distributed hash table, and file exchange in the system. For the data communication module, the link between the PC and the server is connected by TCP/IP.

3.2. Network Diagram

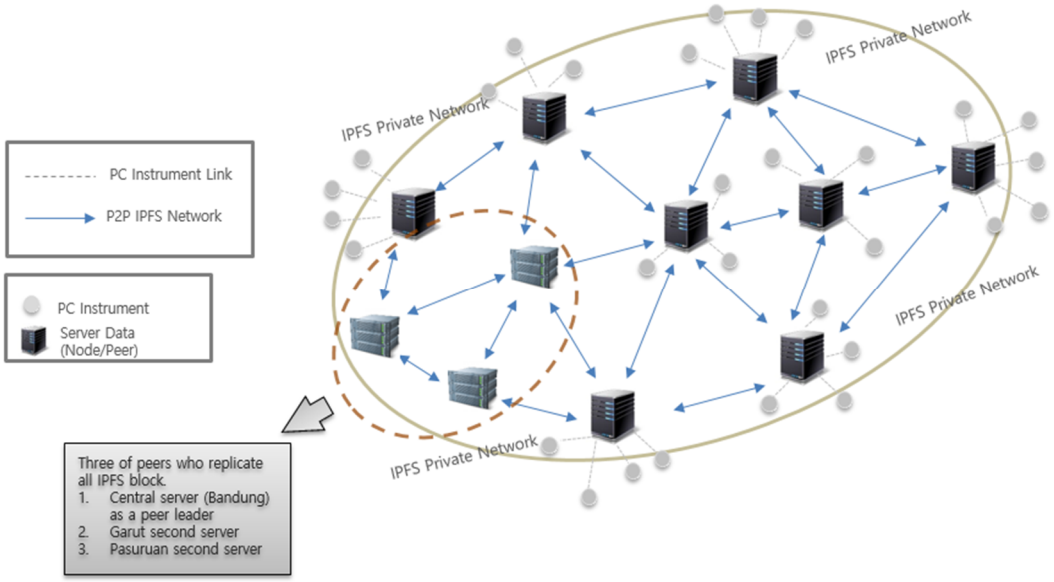


Figure 5. The proposed system architecture.

Our system uses a Virtual Private Network (VPN) to build link communication in our underlying network provided by the vendor. One of the advantages of IPFS is it supports for connection behind the firewall. IPFS is applied to all servers (sites and central) which use an IPFS-Cluster to connect them as members in one cluster. Each server has a unique identifier called a node ID and uses a multiaddr-formatted byte string to communicate among nodes in the overlay network. Multiaddr is used to support addresses for any network protocol and also provides support for encapsulation of the addresses [8]. Additionally, the IPFS-Cluster is used to give the limitation of file distribution flow only in our environment.

Because each site has various instruments that produced data with multiple ranges of time, they are used as the standard peers, and the central server is used as a leader peer in the cluster consensus. When nodes need to exchange files, a peer-to-peer connection is built between them, so a node can connect to another directly (see Figure 5). For example, the node in Biak, which is located in Papua Island, has collected the latest file from instrument CADI, and it is ready for distribution to a node in another island, i.e., node Bandung.

.

Table 1. Observatories status in Q1 of 2019

Observatories	Total Amount of Daily Data (MB)	Bandwidth (Mbps)	Internet Connection Link Type	Frequent of Server Down		
				Network offline (FO Cut)	Power Outages	OS Failures
Garut (Java Island)	21	5	Fiber Optic	0	0	0
Pasuruan (Java Island)	39	7	Fiber Optic	4	0	0
Biak (Papua Island)	46	5	Fiber Optic	1	30	0
Agam (Sumatra Island)	76	4	Wireless	2	0	0
Kupang (Nusa Tenggara Island)	106	5	Wireless	No data	No data	No data
Pontianak (Borneo Island)	110	7	Fiber Optic	3	4	0
Manado	151	1	Fiber Optic	No data	No data	No data

(Sulawesi Island)						
Sumedang (Java Island)	175	4	Fiber Optic	4	0	0

As they are connected in a cluster of peers, the Bandung peer requests that file from Biak and P2P connection is built between them; Bandung can get the data directly from the Biak peer. This scenario is also occurring for all peers, not only Bandung and Biak. Bandung is essential as the main site of our data center and has a significant number of users who need the data, so Bandung must collect all files by replication each time new files are created. However, the proposed system does not replicate files to all peers in consideration of network and storage cost. Each node has distinct characteristics in bandwidth capacity, link connectivity, and the total amount of data passed daily (see Table 1). For our default setting, this system replicates files only to the other three nodes, similar to the replication factor in the Hadoop Distributed File System (HDFS) [19].

According to Table 1, we consider choosing other nodes for storing the data based on the same geographical location with Bandung in Java Island. This will have an impact on reducing the time when Bandung needs to get the file from other nodes which are indicated as offline in their status. Moreover, we consider selecting the other nodes which have the lowest total size in collecting data for daily, nodes with higher bandwidth capacity than others and prefer to choose nodes that are connected by fiber optic links. Thus, based on these considerations, the nodes in Garut and Pasuruan are selected as the second and third center of data replication.

Currently, node Garut has 5 Mbps of bandwidth capacity with a fiber optic connection and is collecting 21 MB of data daily. Meanwhile, node Pasuruan has 7 Mbps for the bandwidth capacity by fiber optic link and is collecting 39 MB of data daily. In Figure 5, our proposed scheme adds the second server in Garut and Pasuruan that has a role of only receiving the data replication from others together with Bandung. This approach is inspired by the hosting role in the Sia project [13].

3.3. Sequence Diagram

In our system, two methods were used for collecting data between machines. First, when the instrument generates a new file and stored in the PC, the data watcher service charges the PC for sending the latest data to the site server. A Secure Shell (SSH) port is used for the PC (having the installed Unix OS) to synchronize the data by Rsync and FTP for the PC which has Windows OS.

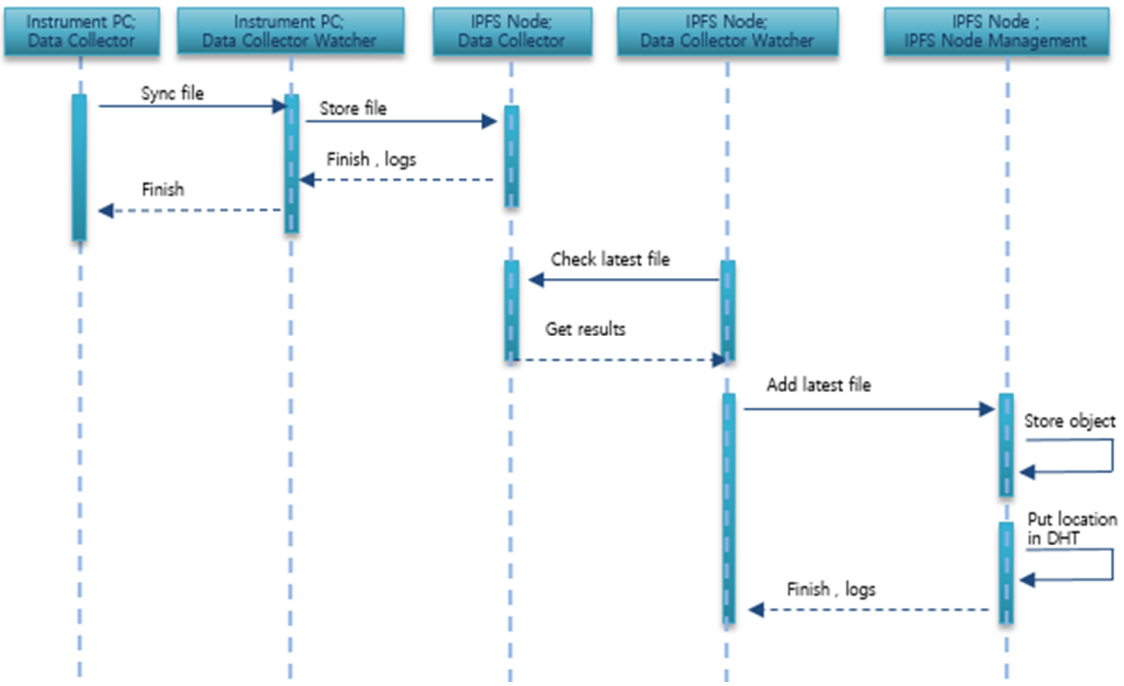


Figure 6. The Sequence of how the file stored.

Secondly, a data watcher was implemented to observe the root directory of data in the site server based on instrument type. Thus, this watcher will force the server to upload the latest file into the IPFS network each time it is created. Afterward, the server, as an IPFS node, will store the data as an object in the local repository and put the metadata of object in the DHT (Figure 6).

3.4. Monitoring Approaches

Monitoring was created on the peer leader to monitor data flow and nodes status. This monitoring is performed by relying on active and passive approaches. Active monitoring is approached by pinging the aimed nodeID in the IPFS network (Figure 7). It runs every minute to obtain status information from all nodes. This ping is not just sending ICMP packets to the destination node but must go through several additional stages.

When the ping command is executed, the node will find the nodeId in DHT first, then establish a connection to that node. Henceforth, it sends the ICMP packet and waits for the pong response. If the destination node is offline, then the pong produces an error; otherwise, it receives a reply and finally prints out round trip time (RTT) for its latency. The pong result is formed to a JavaScript Object Notation (JSON) file and displayed on a web page by a JQuery function. NodeJS was used as our HTTP server, and the index page will update the status every 20 s.

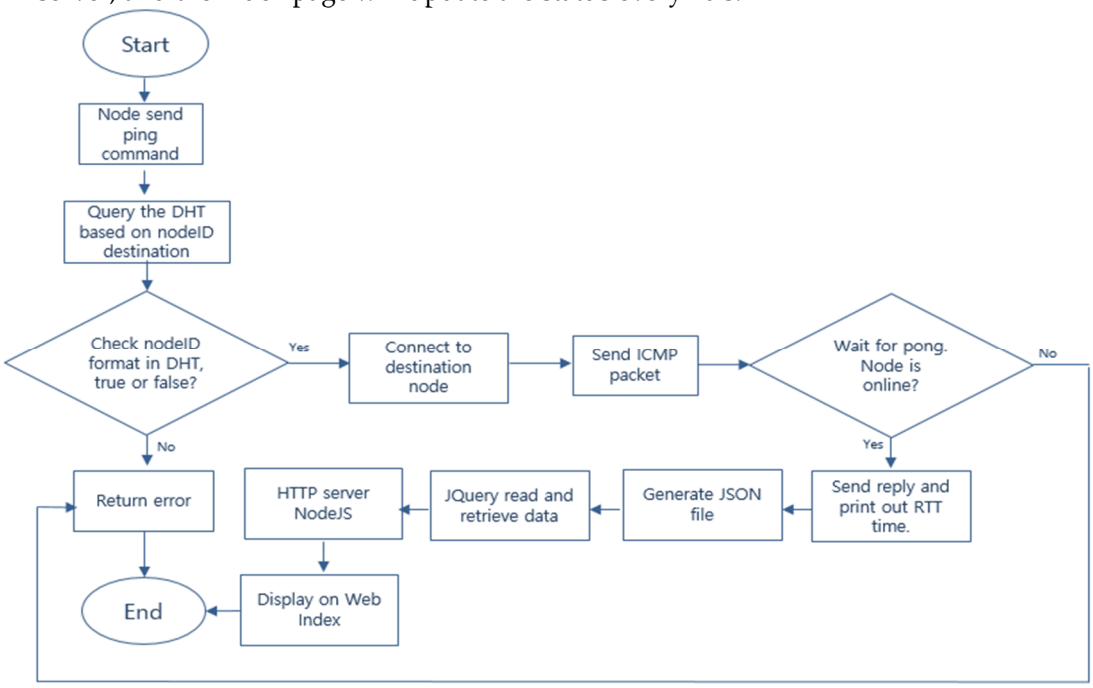


Figure 7. Flowchart of node status monitoring by the active approach.

Moreover, our passive approach utilizes a data collector watcher that monitors each time the latest files are sent from the instrument PC to the site server. It then uploads file to the IPFS via the IPFS-Cluster automatically. This process generates a log file based on the instrument's name that contains information of the metadata file, such as filename, timestamp, file size, block hash, instrument's name, and total size. Additional synchronization of all log files was conducted between the site and central server every minute using Rsync. Updated log files on the main server will be retrieved by JQuery via HTTP and displayed on the website (Figure 8).

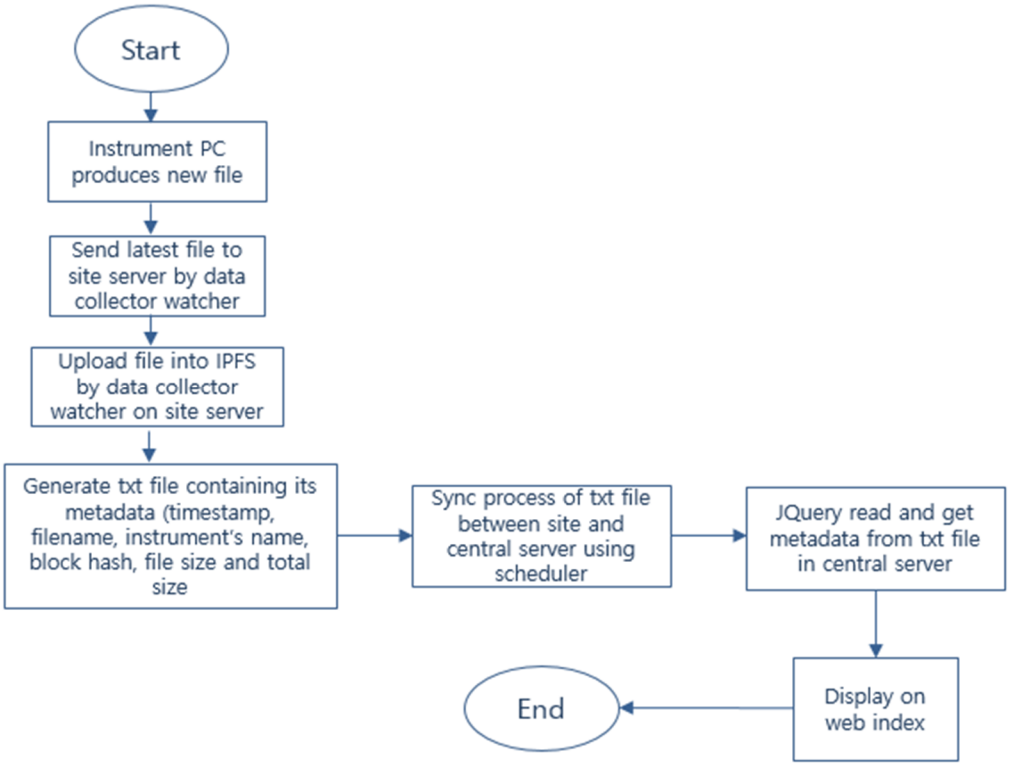


Figure 8. Flowchart of data monitoring by a passive approach.

4. Implementation

This section discusses the detailed implementation of IPFS in our system. We also describe how files distribute among peers and the data structure in the system. In addition, we show a web page of data monitoring and status of nodes.

4.1. File Distribution

The IPFS relies on the BitSwap protocol to exchange blocks between nodes and a distributed hash table (DHT) to store the pointers of peers who have actual locations of files and node information [8, 20] Basically, our proposed system was developed from the existing system and focused only on changing the storage system model. This system implemented the IPFS to increase data availability and provide real-time data for supporting SWIFtS system. In the IPFS configuration, there is a useful parameter, called *storageMax*, to set the upper limit for the size of the IPFS repository that is stored in our local disk. The IPFS-Cluster establishes a priority for block replication based on the maximum free space of the peer. Therefore, setting the configuration of *storageMax* for storing the IPFS repository on the peer leader as the highest level is considered. This peer must store all blocks from all nodes due to having the primary user and being located in the main data center. Next, the configuration of *storageMax* is set by selecting other peers, which collect the lowest size of total daily data considering which peers have a more stable connection. Thus, the system will execute a sequence to replicate them according to those configurations.

A file which is uploaded to IPFS network will map into an object or called block with fixed size using a combination of a hash function and base encoding. The SHA256 hash function has been applied and made the size of each block at most 256 KB because the IPFS uses the Rabin fingerprint

method for chunking files. The Rabin fingerprint method uses content-defined chunking that identifies each fingerprint based on a 20-byte SHA hash value [21] The content calculation is used to create a hash name for each block, so no duplication occurs with the same content stored at the node. Each file that is uploaded to the IPFS has each hash name as a Content Identifier (CID) in the network. The data structure in this system uses a Directed Acyclic Graph (DAG) forming Merkle trees based on the CID that interconnects objects as linked hashes [20].

A file that is added to the IPFS may become one block, or several chunks of blocks, depending on its size. This means that a file smaller than 256 KB would become a block with one unique hash name that has the size like its original.

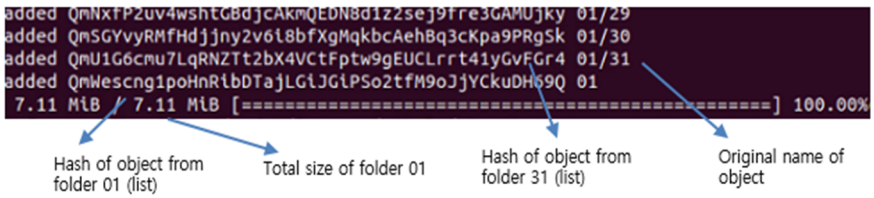


Figure 9. Upload CADI data containing files less than 256 KB.

In Figure 9, we can see the example of upload the January 2019 data (folder 01) that includes sequence folder of date from 01-31 and each of date folder containing files with size less than 256 KB. Each folder has a distinct hash name, and a hash link connects them. Meanwhile, in figure 10, we can see if a block has size more than 256 KB, it is broken into several chunks of the block according to its total size. Data was uploaded from a January folder (Jan.) from the SN-4000 instrument containing the folders dated from Jan. 1 to Jan. 31; each date's folder also includes files with sizes greater than 256 KB. All blocks which derive from the root of the block have a different hash name linked as a Merkle tree of the block (Figure 11).

Figure 10. Uploaded SN-4000 data containing files greater than 256 KB.

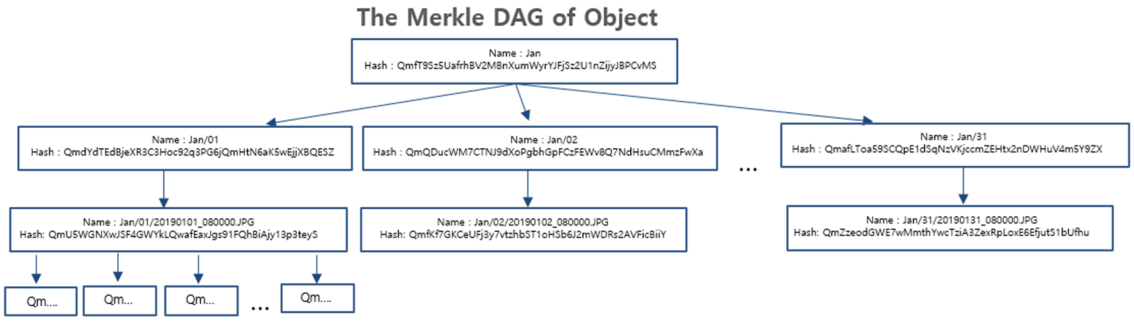


Figure 11. The Merkle link of folder Jan from the SN-4000 Instrument.

In logical format, the IPFS Object data structure has two fields, Link and Data. Link contains an array of link structures (links to other objects/blocks) whereas Data is a blob of unstructured binary data (data in the block). In the Link structure, there are three data fields containing important information such as Name, Hash, and Size. Name field includes a name of the link while Hash is a hash of link of the object/block. Then, the Size field containing a cumulative size of the object/block. Figure 10 depicts an example of the hash of a file that has a size of 2.2 MB. If we investigate its data structure, the Link part of that file contains all chunks of block hashes that are connected to the root hash with a maximum size of 256 KB (Figure 12).

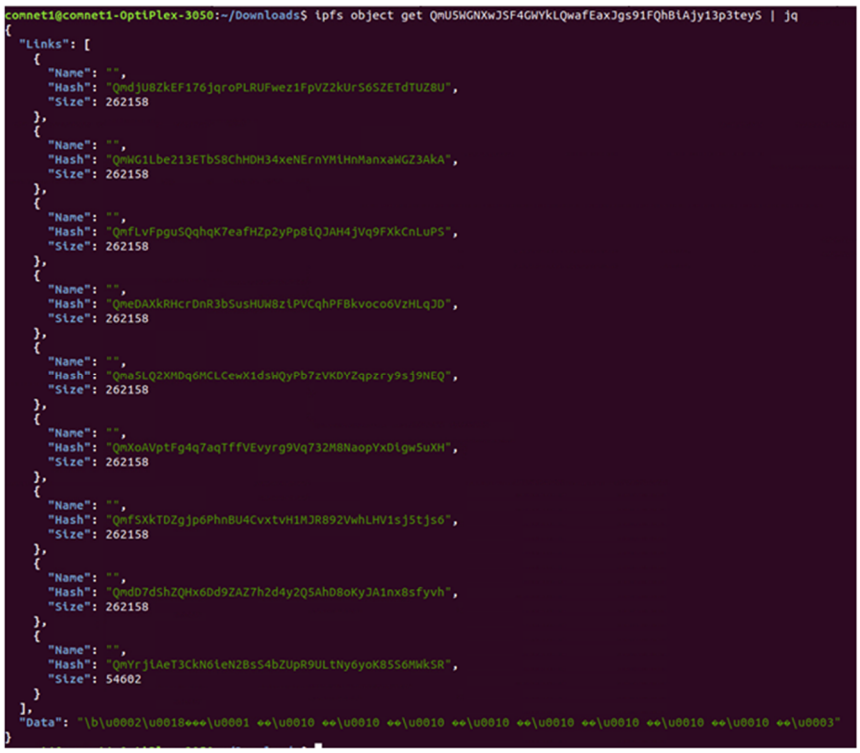


Figure 13. Hash links from one block that has size 2.2 MB (filename: 20190101_080000.JPG).

This system relies on the DHT and BitSwap protocol for the block exchange process. The DHT provides two essential functions [22]:

1). Peer routing; used to identify other peers on the network based on PeerID (nodeID). This information only lists the online node to be announced to others.

2). Content routing; serves information about data published by nodes and records file locations.

When a peer queries the DHT based on a requested hash, the system will check the DHT to determine whether the requested hash is referenced or not. If not, these peers will get the data directly from DHT, and the system will update the DHT. Otherwise, the DHT will route the requester and connect to the peer that stored the hash (provider) to obtain the hash from the provider. Finally, the system stores new records in the DHT.

The centralized exchange of blocks on this system utilizes the BitSwap protocol. This protocol is a message-based protocol responsible for exchanging blocks between peers and handles the requests to fetch data blocks from the network. BitSwap manages two processes; the first acquires a set of blocks (*want_list*) that have been requested by a peer and the second sends a set of blocks (*have_list*) that the peer requests. After DHT assists a peer in finding another peer that has the block, a P2P connection is built, and that peer (sender) sends a message to request the block as its *want_list*. Another peer who listens to the request (listener) will check its *have_list* for the block. If there is a match between the *want_list* and the *have_list*, the block is sent by the listener to the sender directly. BitSwap Ledger records this transaction, and the P2P connection is closed. Otherwise, if there is no match between sender and listener, no transaction occurs, no new transaction is recorded in the BitSwap Ledger, and the P2P connection closed.

4.2. Monitoring Web Interface

A web interface has been implemented for data and node status monitoring in the system. This simple index page is built with the Bootstrap 3.3 framework consisting of HTML, CSS, and JS. JQuery was also used to read and retrieve data from daily log files as well as for automatic refreshing of the page every 20 s. For data transfer, NodeJS version 10.15.3 was implemented as the HTTP server. NodeJS handles data transfer when JQuery extracts the data from the files and communicates it to the HTML page. Each node has several log files according to its instruments. Therefore, separate JQuery functions are used for reading and extracting the data from those files; one function is

responsible only for one file. In Figure 14, we can see that this monitoring is displayed on only one page of the index that contains information on the data that has been successfully stored in the IPFS network and distributed to other nodes. In addition, the status of each node is also displayed to determine the current state of the nodes.

The log files automatically update every minute. For data monitoring, logs were sent from all sites to the central server. Meanwhile, for node status monitoring, status information comes from files generated by the central server after finishing pings. Figure 14 presents data monitoring per node and displays the status of nodes on the bottom. Nodes 2 and 3 succeed in showing their data because their state is online, contrary to Node 4, which has no data information due to its offline status.

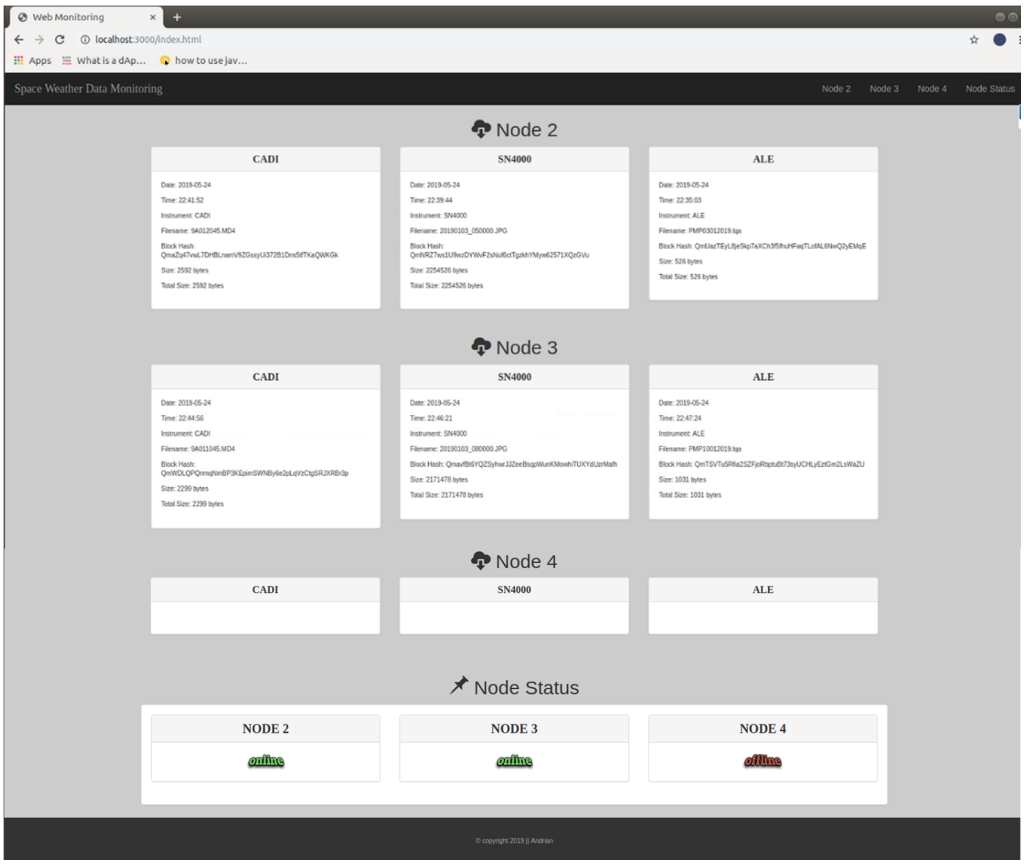


Figure 14. Index page for web monitoring.

5. Experiment

This section discusses the experiment that has been conducted to determine the performance of the proposed system compared to the existing system by calculating mean replication time and the mean throughput of a node.

5.1. Experimental Data and Method

This experiment is intended to evaluate the Rsync that has been implemented by the existing system compared to the IPFS coupled with the IPFS-Cluster. The tests were conducted using two PC machines with the following specifications:

Table 2. Specification of Tested Machines

Computer 1	
Processor	Intel(R) CoreTM i5-7500T CPU @2.70 GHz
RAM	8 GB
HDD	250 GB ATA
OS	Ubuntu 18.04
Computer 2	
Processor	Intel(R) CoreTM i5-3317U CPU @1.70 GHz
RAM	4 GB
HDD	500 GB ATA
OS	Ubuntu 18.04

Several experiments with tested parameters were conducted at the computer network lab of Keimyung University. Both Computer 1 and Computer 2 have the same role that represents servers in the system. Each machine uses a different IP segment address. Computer 1 is set using a public IP connected with a UTP cable, while Computer 2 uses a local IP that is connected to Wi-Fi. To observe a more realistic scenario, the environment was set as given below:

1). One folder was created containing four instrument folders, namely CADI, SN4000, ALE and FLUXGATE.

2). Each instrument folder contains a year/month/day folder.

3). all files were spread evenly only at folder day "01" of the month "01" (folder 2019/01/01/) on each instrument folder.

4). As for the parameters, the number of files (100) was varied with distinct sizes (256 KB, 1 MB, and 100 MB). The IPFS could not store files with the same content; therefore all data was generated randomly by the /dev/urandom interface for all tests, including Rsync. There were 100 files of each size with five trials for both Rsync and the IPFS + IPFS-Cluster. A total of 9000 datasets were tested for a total of 33,768 GB for its overall size.

5). The network latency between computer 1 and Computer 2 was set such at no latency (default), 5 ms, and 20 ms latency. Latencies are emulated artificially using the Linux Traffic Control (*tc*) tool.

The purpose of our measurements is to determine the performance of the system. The file replication time was measured between computer 1 and computer 2 for both Rsync and the IPFS coupled with the IPFS-Cluster. For Rsync, the duration of a file copy received by another computer was obtained. Meanwhile, in the IPFS + IPFS-Cluster, computer 1 was set as a peer 1, and computer 2 was set as a peer 2 in one cluster. Files are replicated from computer 2 to computer 1. Then, the elapsed time between the "pins" of the file by the peers was computed.

In addition, the mean throughput was derived by calculating the average of the data replicated per second; this can be obtained by dividing the total file size by the time elapsed during replication.

5.2. Mean Time of Files Replication

Overall, the results show that replication using the IPFS + IPFS-Cluster is faster than Rsync. This is because the IPFS replicates all files using chunks formed by the hash function which have a small and fixed size.

Figure 15 presents the replication time of files as a function of latency when all files are only replicated from computer 2 to computer 1. The main observation is that the latency has a more significant impact when replicating large files. File size determines the duration of replication; the larger of file size, the longer it will take to replicate the file. The effect of latency also appears to be very significant on the replication time. It can be seen that high latency makes a considerable difference compared to low latency. It takes 297.64 s to replicate files of 1 GB (100 x 10 MB) when the default latency is considered, and 448.12 s when the latency of 20 ms is used for the IPFS+IPFS-

Cluster. This happened similarly to Rsync, which was 336.36 s at the default latency and 469.05 s when the latency was 20 ms.

Another interesting observation is that file replication by Rsync experiences a slight drop within small and large latencies where it takes 13.86 s and 12.8 s for file sizes of 25.6 MB (100 x 256 KB) and 38.22 s and 38.19 s, respectively, with a size of 100 MB (100 x 1 MB). This might be due to the more stable bandwidth used or no usage sharing Wi-Fi with other users. On the other hand, file replication by the IPFS + IPFS-Cluster has a rising trend when latency increases. Replication takes 8.95 s to 9.66 s for the file size of 25.6 MB (100 x 256 kB) and 36.53 s to 45.41 s with the file size of 100 MB (100 x 1 MB).

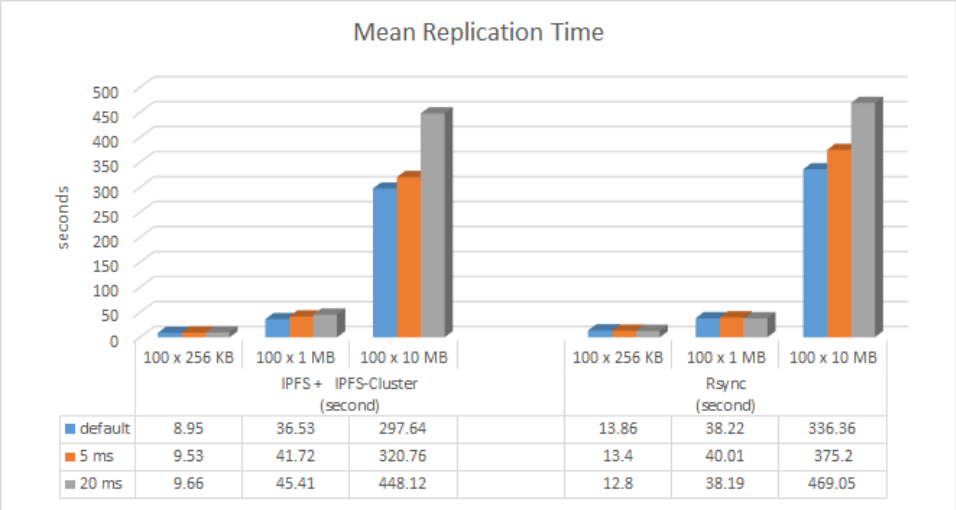


Figure 15. Mean of time to replicate files with a size of {25.6 MB, 100 MB, 1 GB} as a function of latency.

5.3. Mean Throughput

The mean node throughput is derived by dividing the average size of the replicated file by the transfer time. Figure 16 shows the mean throughput based on the number of files that are replicated from computer 2 to computer 1 with either the IPFS + IPFS-Cluster or Rsync. In general, the results show that the throughput by the IPFS + IPFS-cluster is greater than that of Rsync for each file size. The main observation is that the file size affects the level of throughput produced. Where the highest throughput generated is 3,359,763 B/s for the IPFS + IPFS-Cluster and 2,973,005 B/s for Rsync for a 1 GB (100 x 10 MB) file size. Meanwhile, a slight decrease in throughput of approximately 243,904 B/s occurs in the IPFS + IPFS-Cluster when replicating files with a size of 25.6 MB (100 x 256 KB) and 100 MB (100 x 1 MB). This is in contrast to the throughput generated by Rsync, which shows an increasing trend of approximately 18% to 35% for all tested files.

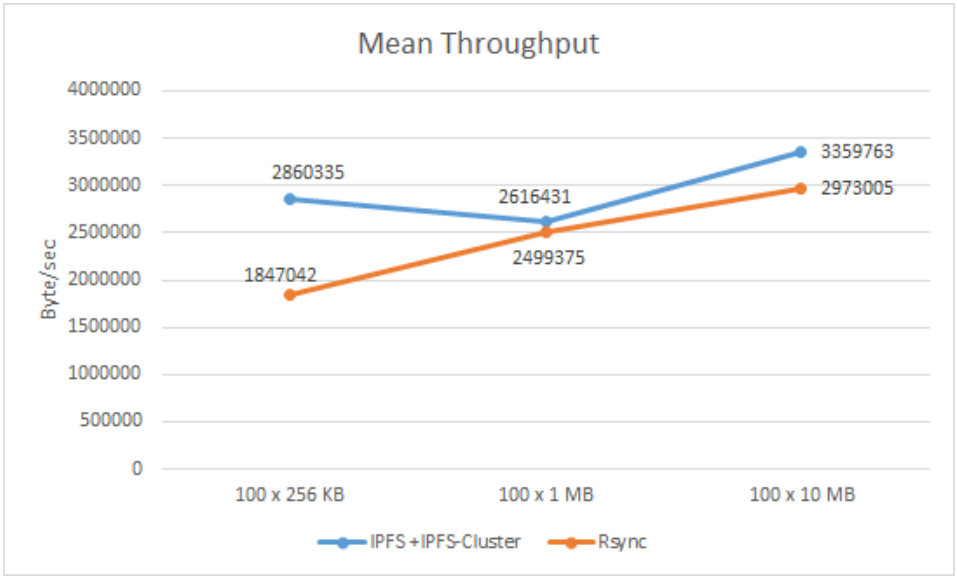


Figure 16. Mean throughput of a node

The conduct of the network bandwidth measurement was supported by the Iperf for deriving the average network bandwidth between computer 2 and computer 1. Figure 17 shows the measured bandwidth with *Iperf* between computer 2 (client) and computer 1 (server). The average bandwidth of the measurement was calculated at 26.4 Mbits/s. If we compare the highest node throughput score of the IPFS + IPFS-cluster with that of Rsync, it is seen that the IPFS + IPFS-Cluster throughput meets the average throughput generated by the network with a rating (3,359,763 B/s * 8)/1,000,000 = 26.8 Mbits/s. This indicates that the IPFS-IPFS-Cluster supports high-throughput so that it reduces the time consumed by file distribution.

```
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 210.125.[redacted] port 5001 connected with 210.125.[redacted] port 47116
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.4 sec  34.8 MBytes  28.0 Mbits/sec
[  4] local 210.125.[redacted] port 5001 connected with 210.125.[redacted] port 47120
[  4]  0.0-10.6 sec  32.1 MBytes  25.5 Mbits/sec
[  4] local 210.125.[redacted] port 5001 connected with 210.125.[redacted] port 47122
[  4]  0.0-10.6 sec  33.5 MBytes  26.5 Mbits/sec
[  4] local 210.125.[redacted] port 5001 connected with 210.125.[redacted] port 47166
[  4]  0.0-10.3 sec  31.9 MBytes  25.9 Mbits/sec
[  4] local 210.125.[redacted] port 5001 connected with 210.125.[redacted] port 47168
[  4]  0.0-10.5 sec  32.8 MBytes  26.1 Mbits/sec
```

Figure 17. *Iperf* measurement results

6. Conclusion

Today, the growth of space weather data is rapidly demanding data availability to accommodate the needs of researchers. This thesis proposed a new system to increase data availability for supporting SWIFtS using a decentralized storage method. Our proposal leverages the IPFS network to distribute and store data on the IPFS node. A directory watcher is added, forcing both instrument PCs to synchronize data and nodes to upload data automatically. Moreover, the implementation of the IPFS-Cluster is useful to replicate the data between peers as well as to limit the distribution process only in cluster peer members.

Furthermore, a combination technique for system monitoring has been implemented and is seen to be useful in providing real-time data flow and node status. On the evaluation of the finished work, aiming to learn the proposed system performance compared with the existing system, the evaluation has shown that the IPFS-based solution is able to reduce the time of file replication and support high-throughput.

Future work is needed to improve the speed of data distribution among peers and include a more extensive exploration of the hardware effect on system performance. Moreover, traffic monitoring of bandwidth usage is also an important approach to better understand the network cost from this system.

Acknowledgements: This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00539, Development of Blockchain Transaction Monitoring and Analysis Technology)

Conflict of Interest: The authors declare no conflict of interest

References

1. LAPAN. Available online: https://lapan.go.id (accessed on 2 January 2019).
2. SWIFtS. Available online: http://swifts.sains.lapan.go.id (accessed on 2 January 2019).
3. Wang, S.; Zhang, Y.; Zhang, Y. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *IEEE Access* **2019**, 6, 38437–38450. [CrossRef]
4. Ali, S.; Wang, G.; White, B.; Cottrell, R.L. A blockchain-based decentralized data storage and access framework for PingER. In Proceedings of 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), New York, NY, USA, 1-3 August 2018; pp. 1303-1308. [CrossRef]
5. Kuo, T. T.; Kim, H. E.; Ohno-Machado, L. Blockchain distributed ledger technologies for biomedical and health care applications. *Int. J. of the American Medical Informatics Association*, 2017, 6, 1211-1220. [CrossRef]
6. Shafagh, H.; Burkhalter, L.; Hithnawi, A.; Duquennoy. S. Towards blockchain-based auditable storage and sharing of IOT data. In Proceedings of 2017 on Cloud Computing Security Workshop, ACM, Dallas, Texas, USA, 3 November 2017; pp. 45–50. [CrossRef]
7. Wennergren, O.; Vidhall, M.; Sörensen, J.; Steinhauer, J. Transparency Analysis of Distributed File Systems. Bachelor Degree Project in Information Technology, University of Skövde, Sweden, 2018.
8. Benet, J. IPFS-content addressed, versioned, P2P file system. Preprint arXiv: 1407.3561, 2014.
9. Confais, B.; Lebre, A.; Parrein, B. An object store service for a Fog/Edge Computing infrastructure based on IPFS and a scale-out NAS. In Proceedings of Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference, Madrid, Spain, 14-15 May 2017. [CrossRef]
10. Jovović, I.; Husnjak, S.; Forenbacher, I.; Maček, S. 5G, Blockchain, and IPFS: A General Survey with Possible Innovative Applications in Industry 4.0. In Proceedings of 3rd EAI International Conference on Management of Manufacturing Systems-MMS, Dubrovnik, Croatia, 6-8 November 2018; pp. 1-10. [CrossRef]
11. De la Vega, F.; Soriano, J.; Jimenez, M.; Lizcano, D. A Peer-to-Peer Architecture for Distributed Data Monetization in Fog Computing Scenarios. *Int. J. of Wireless Communications and Mobile Computing*. 2018, 1-15. [CrossRef]
12. Kroes, N. Digital Agenda and Open Data. Available online: https://europa.eu/rapid/press-release_SPEECH-12-149_en.htm (accessed on 11 January 2019).
13. Vorick, D.; Champine, L. Sia: Simple decentralized storage. Available online: https://sia.tech/sia.pdf (accessed on 31 January 2019).
14. Wilkinson, S.; Boshevski, T.; Brandoff, J.; Buterin, V. Storj: A Peer-to-Peer Cloud Storage Network. Available online: https://storj.io/storj.pdf (accessed on 2 February 2019).
15. Lambert, N.; Ma, Q.; Irvine, D. Safecoin: The Decentralized Storage Token. Available online: https://docs.maidsafe.net/Whitepapers/pdf/Safecoin.pdf (accessed on 6 February 2019).
16. Nygaard, R. Distributed Storage with Strong Data Integrity based on Blockchain Mechanisms. Master's thesis, University of Stavanger, Stavanger, Norway, 2018.
17. Brisbane, S. Decentralising Big Data Processing. Bachelor thesis, The University of New South

Wales, Sydney New South Wales, Australia, 2016.
18. IPFS-Cluster. Available online: https://cluster.ipfs.io/ (accessed on 5 December 2019).
19. HDFS Architecture Guide. Available online: https://docs.huihoo.com/apache/hadoop/1.0.4/hdfs_design.pdf (accessed on 15 February 2019)
20. Interplanetary Linked Data (IPLD). Available online: https://ipld.io/ accessed on 28 December 2018).
21. Kaiser, J.; Meister, D.; Brinkmann, A.; Effert, S. Design of an exact data deduplication cluster. In Proceedings of IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, CA, USA, 16-20 April 2012; pp. 1-12. [CrossRef]
22. IPFS Architecture. Available online: https://github.com/ipfs/specs/tree/master/architecture (accessed on 10 January 2019).