Summer 6-22-2021

Improving the Security and Performance of Web Applications Running on the Distributed IPFS

Vu Le

San Jose State University

Improving Security and Performance of Distributed IPFS-based Web Applications with Blockchain

A Project Presented to
The Faculty of the Department of Computer Science
San Jose State University

In Partial Fulfillment of
the Requirements for the Degree
Master of Science

By
Vu Le
May 2021

SAN JOSE STATE UNIVERSITY

The Designated Project Committee Approves the Master's Project Titled

Improving Security and Performance of Distributed IPFS-based Web Applications with Blockchain

By

Vu Le

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Melody Moh	Department of Computer Science
Dr. Ramin Moazeni	Department of Computer Science
Dr. Teng Moh	Department of Computer Science

ABSTRACT

While cloud computing is gaining widespread adoption these days, some challenges are emerging around security, performance, and reliability of centralized cloud resources. Decentralized services are introduced as an effective way to overcome the limitations of cloud services. Blockchain technology with its associated decentralization is used to develop decentralized application platforms. The interplanetary file system (IPFS) is built on top of a distributed system consisting of a group of nodes that shares the data and also takes advantage of blockchain to permanently store the data. The IPFS is very useful in transferring data between people. This project focuses on blockchain technology, distributed file sharing system IPFS, and their applications in software development. It talks about different types of blockchain, its advantages and challenges, and what we can do to overcome the challenges of decentralized technology.

Keywords **- blockchain technology, distributed system, decentralized service, interplanetary file system (IPFS)**

ACKNOWLEDGEMENT

I would like to thank my project advisor, Dr. Ramin Moazeni, for his support and guidance throughout the project. I also want to thank Dr. Melody Moh and Dr. Teng Moh for the support and being my committee members. Finally, I am thankful for the support of my family to help me achieve my academic goal.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Cloud computing is widely used in software applications. It uses centralized computing resources on remote cloud servers, such as Amazon AWS and Google Cloud. A 2018 survey on enterprise cloud computing found that 73% of organizations use cloud infrastructure in at least one of their applications, and 17% intend to use cloud resources in the next 12 months. However, there are certain limitations with using centralized cloud resources because the cloud can be a single point of failure, as shown by many such examples in the past. Take, for example, a careless mistake committed by a cloud operator can cause the cloud, and all the companies and organizations relying on the cloud resource, essentially half of a country, to shut down for up to an entire day. Similarly, a power outage, a data loss, or a network attack on these services can potentially disrupt the operations, causing huge damages to user data and business activities. A survey was conducted by RightScale in 2018; it asked 997 technical professionals about the challenges of adopting cloud technology. The survey showed some interesting findings. Security was the top concern mentioned by 77% of them, and 55% of them faced performance challenges while using cloud technology [3].

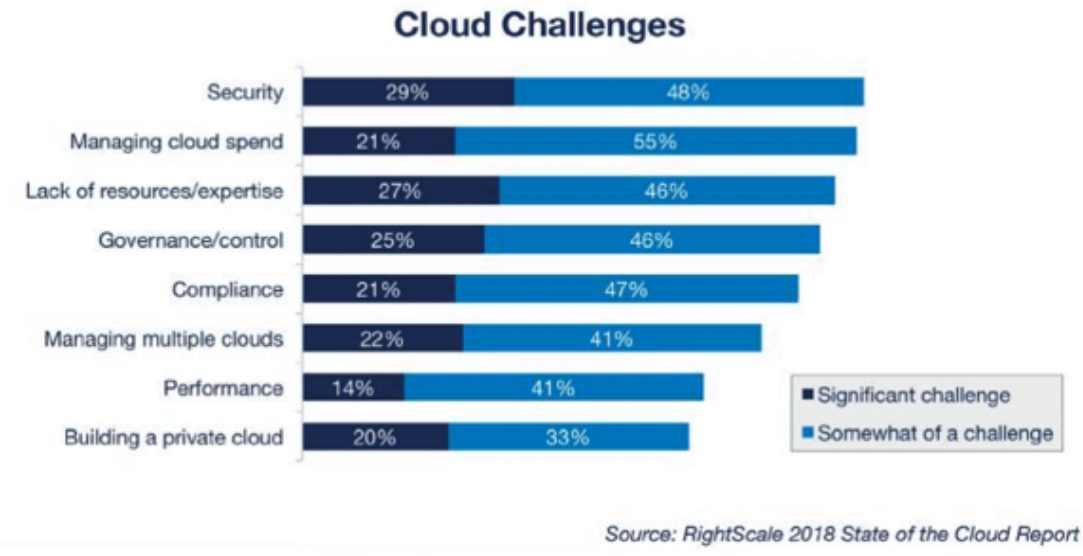

Figure 1. Cloud challenges faced by technical professionals

In this paper, we address the security and performance challenges of distributed systems. There are several blockchain-based solutions to mitigate the drawbacks of centralized cloud resources. The computing resources need to be decentralized and performed by multiple parties, so blockchain processes the right characteristics and functionalities as a blockchain is stored on a ledger that is shared between all participating nodes on a network. In addition, any transaction that takes place on the blockchain requires consensus of all the nodes, so it is also very secure. It is important that there is no single entity which controls the blockchain, so it does not have the single point of failure issue as in other regular cloud storages.

A new file system called interplanetary file system (IPFS) has been invented and can process and store large files in a decentralized and effective manner. IPFS runs on a peer-to-peer network. It has decentralized storage and possibly encryption on each node in the network to avoid a single point of failure issue in centralized storage. It helps prevent data losses since the same data is available on multiple peer nodes [11]. Based on the above reasons, IPFS combined with blockchain technology can serve as a great solution to address the security and performance concerns of cloud technology.

This work describes the design of applying blockchain to IPFS to support secured distributed web applications, including applying smart contracts for access control [16]. The detailed implementation is depicted, including encryption/decryption schemes, smart contracts, remote application migration, and memory cache for key values. Performance and security analyses are presented, with a discussion of tradeoffs between the two.

In this project, we have studied published research papers and online articles to come up with a number of ideas we can integrate into our application to make it more secure, available, and

performant. The rest of the paper is organized as follows. After the introduction, the next section II will go over preliminary background information and related work about blockchain technology, smart contracts on blockchain, IPFS, distributed hash table, and memory cache. Section III demonstrates the high level architectural design of the web application and the design of each component. Section IV talks about and analyzes our implementation approaches, the actual implementation methods, and how everything works together in the application. In Section V, we detail our experiments and their results. Finally, section VI is the conclusion and future work.

II. BACKGROUND AND RELATED WORK

2.1 Blockchain Technology

Blockchain technology is an open source and distributed system that enables transactions between two parties in a secure, verifiable, and trusted way, as illustrated in Dragonchain [2]. The underlying system behind blockchain is a distributed peer-to-peer network. There is no centralized authority that controls the network. Each node in the network performs its work independently. A new node needs to be verified by existing nodes before it joins the blockchain network. All the nodes are required to reach a consensus on a transaction before the transaction is recorded on a ledger that is shared among all the participating nodes. A consensus is made by each participant solving a complex puzzle and providing a proof of work or proof of stake to a server. The server checks the proof from each participant and rewards the ones with the correct proofs [1]. Once the proof has been approved by the server, the transaction is confirmed, and a new block which contains the hash code and the transaction is added to the ledger, as illustrated by Tar [17]. The new block creation process is called mining. A node gets rewarded with money for successfully mining a block first, so all the competing nodes race to process a transaction and create a new block; refer to Salman et al. [13]. This approach improves the security of the blockchain network. Data on the blockchain is immutable, so it will never change after it is written to the ledger. One limitation of blockchain is that it relies on continuous communications between nodes in the network, so its performance will be largely impacted in areas with a low bandwidth or slow connection, explained by Hu et al. [5].

Note that although blockchain is a great solution for decentralized computing resources, storing data on a blockchain alone is expensive because the same data needs to be propagated to

and processed on other nodes in a blockchain network. As there is a large amount of data on the Internet now and getting larger in the future, a node will need a more powerful CPU, higher bus speed, higher bandwidth, and a bigger storage to process all the data, which will increase the cost to operate blockchain. There are three different types of blockchain: public blockchain, private blockchain, and hybrid blockchain.

A. Public blockchain

A public blockchain is open source and fully decentralized. Any node is allowed to participate in. All transactions taking place in a public blockchain are transparent and can be examined by any participating node. The nodes do not know each other. A public blockchain also maintains a token to incentivize participants [1, 2].

B. Private blockchain

In contrast to a public blockchain, a private blockchain is not open to the public and requires permission for a new node to join. Transactions happening in a private blockchain are only visible to the group of nodes who have been granted permission to join the network [2]. A private blockchain is more centralized because it is internally managed by a controlling entity. The participating nodes know each other. A private blockchain may or may not have a token to incentivize participants. It is more suitable for use in an enterprise where their sensitive data should be protected from being exposed to the public. A subtype of private blockchain is called a consortium blockchain. Instead of being governed by one entity, it is controlled by a group of entities. Participants in a consortium blockchain include business partners or competing businesses as long as they can collaborate on some aspects of their business. It enables them to work effectively to reach a shared goal [1].

C. Hybrid blockchain

Besides public and private blockchains, a hybrid blockchain is a combination of both public and private blockchains. It offers both the transparency and security of a public blockchain and the privacy of a private blockchain. A public blockchain can connect to a private blockchain, creating a multi-chain network of blockchain. It also increases the security of a transaction since it will be processed and encrypted by more than one blockchain. Another difference is that a hybrid blockchain does not need proof of work or proof of stake protocol [2]. This model is applicable to businesses that want to make only some of their data public while keeping other data private.

2.2 Smart Contracts

In order to protect the privacy of users' data on the network, Rahulamathavan et al. (2017) proposed an attribute-based encryption method. The data owner creates user groups and assigns an access policy that consists of certain user attributes to each group. A user must possess all the specified attributes that satisfy the policy to access the data [12]. This encryption technology does not look at users' identity and does not work in a decentralized network as it requires the centralized access policy server. Smart contracts can solve this problem by their ability to work on a decentralized network and customized implementation logic for user verification in every transaction.

Smart contracts are computer programs that are used to execute blockchain based transactions. Smart contracts contain executable code written in Solidity. Solidity is a statically typed language that makes use of variables, functions, and structures. The code follows a mutual agreement between two parties involved in a transaction. Contracts are compiled and deployed to an environment which is capable of running the contracts whenever a transaction takes place. As an example, for Ethereum blockchain, smart contracts are deployed and executed in the

Ethereum virtual machine (EVM) [9]. The code in a smart contract is publicly available on the ledger in a blockchain, so a new node which joins the blockchain later will receive a copy of the smart contract. There is no centralized authority controlling smart contracts, but instead they are distributed across multiple nodes on the network. It ensures trust and anonymity of any smart contract triggered operations. Before a smart contract executes a transaction, a transaction fee is required and, in the Ethereum case, the fee is paid in ether, also known as gas. The transaction fee is higher for smart contracts with more complex logic.

Smart contracts can run 2 types of transactions:

- Ensure the receipt of funds when a transaction takes place
- Give financial penalties if certain conditions are not met, as agreed upon by the parties involved.

The usage of smart contracts has been previously proposed for access control between smart devices in the Internet of Things (IoT). Song et al. (2019) comes up with an attribute-based access control scheme that checks device attribute information and decides whether to allow a device to join an IoT network. This scheme uses blockchain and smart contracts to store and manage access control policies [15]. In addition, Nakamura et al. (2020) proposes a capability-based access control using smart contracts in the IoT. In this scheme, the author suggests having smart contracts manage capability tokens. There is a token created for each capability or action, and smart contracts will verify the ownership and validity of the token from the access requestor to grant them access or not [10].

2.3 Interplanetary File System (IPFS)

Interplanetary file system is a distributed file system where we can store any computer file or data for later access. Unlike a file system on a computer, IPFS divides a file into multiple chunks

of data and distributes the chunks across different nodes on IPFS. While other file systems use a location based address to look up a file, IPFS uses a content based address where a file's address is based on the hash of the file's content. A single unit of data on IPFS is a block. IPFS constructs a directed acyclic graph to link all the blocks in a file, which helps facilitate data retrieval. IPFS also maintains a distributed hash table to know where to find the data or the path which leads closer to the data [7].

When a node downloads data from IPFS, the node can store the data locally and will serve as a new provider of the data. Therefore, when the node's nearby computer requests the data, the node can send the data to the requesting computer, just as any other nodes on IPFS can do.

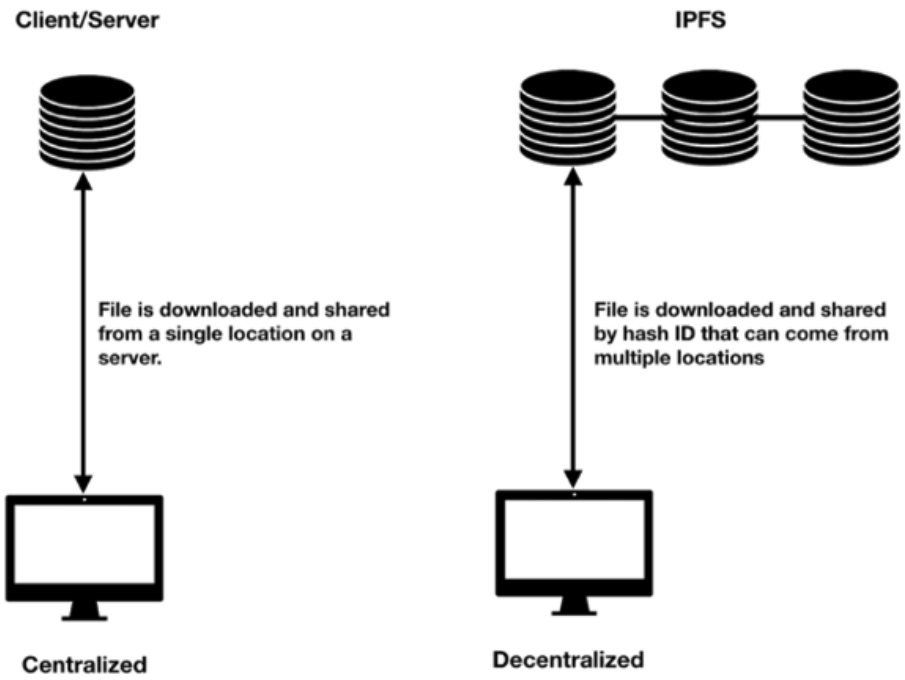

Figure 2. Difference between a centralized storage and IPFS

The paper Shen et al. (2019) analyzes the I/O performance of IPFS and notices a bottleneck in IPFS data retrieval because of the resolving and downloading operations. When a node in IPFS receives a data retrieval request, if the data is not local on the node, it needs to perform a

lookup to find the storage node of the data. If the data size to download is larger than 256KB, IPFS will break it into multiple blocks of 256KB each. When all the blocks arrive at the requesting node, they will be re-aggregated to a file of the original size, so the block size also affects the I/O performance of IPFS [14].

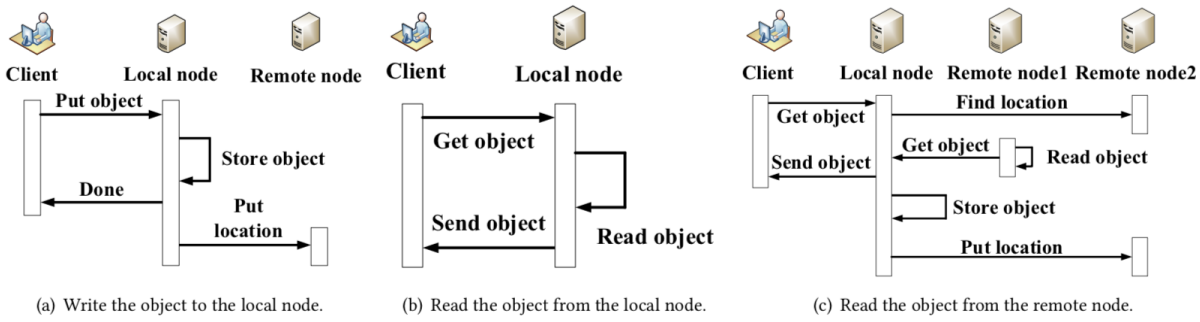

Figure 3. The read and write operations of IPFS

2.4 Distributed Hash Table

A distributed hash table stores key-value mappings to help IPFS find a peer node having the requested data or find a path leading to a node. Each node on the blockchain maintains its distributed hash table and uses it to find and return the value when requested for a key. A key can be a data unique identifier, an Interplanetary Name System (IPNS) key, or a peer ID.

There are 3 types of records in a distributed hash table in IPFS.

Table 1: Types of Records in IPFS Distributed Hash Table

Type	Purpose
Provider records	Map a data identifier with a peer ID that has the data
IPNS records	Map an IPNS key with an IPNS record
Peer records	Map a peer ID with a set of close peers that can reach the peer ID

In the paper, Hassanzadeh-Nazarabadi et al. (2019), the author used the distributed hash table concept to design a blockchain architecture called LightChain. LightChain is a permissionless blockchain where any node can join the network. Any new transaction or block addition to this blockchain will be recorded in the distributed hash table of a peer in an on-demand manner, so each peer does not have to store the entire blockchain locally [4]. The distributed hash table provides efficient data lookup, but it may take a noticeable latency when it requires many table lookups to retrieve the data.

2.5 Caching

Caching is a mechanism to expedite data retrieval when the same data is requested multiple times within a specified recent amount of time during which the data is on the cache. The first request to get the data will trigger usual network and remote API calls to retrieve the data. The retrieved data will be cached in memory on the server, so that subsequent requests to get the data will query the cache and directly get the data from the cache. It saves a lot of network and remote API calls, so the requested data can be retrieved much faster.

III. APPLICATION DESIGN

In this semester, we continued building and integrating more improvements into our file upload and download application, focusing on the distributed file system IPFS and smart contracts on blockchain. We added encryption/decryption, access control in smart contracts, migration of the application to completely run on IPFS, and memory cache to enhance the security and performance of the application.

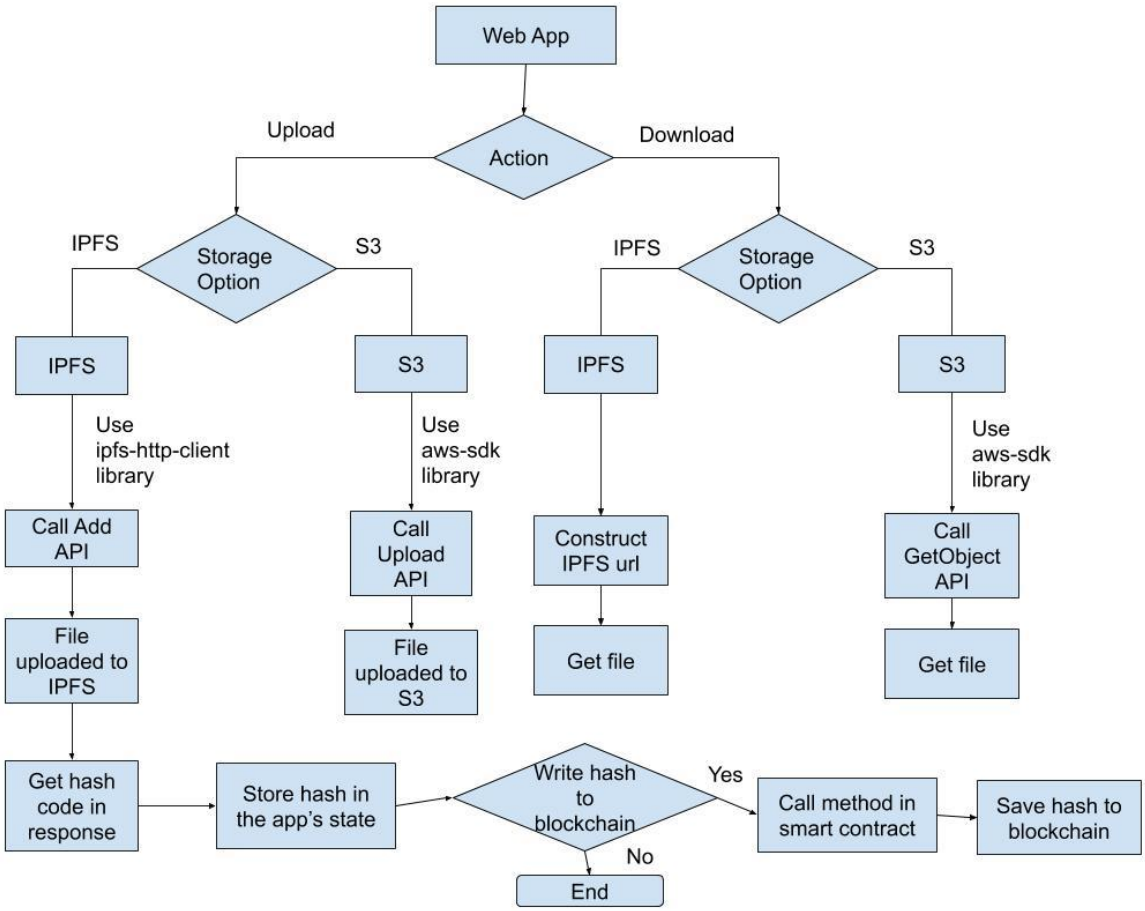

Figure 4. Flow chart for upload and download operations

We are using an IPFS Javascript client to interact with IPFS. Starting with file upload, it is simply taking a file from the user and calling the Add API from the IPFS client. Depending on

the size of the file, IPFS can break the file into multiple blocks of data bytes. The blocks can be stored on different nodes on IPFS. For each file, IPFS creates a directed acyclic graph to link all the blocks together. In addition, IPFS maintains a distributed hash table to keep track of which node has a block of data, so when a file is requested by the user, IPFS knows where to get all the data blocks to construct the file to send to the user. Instead of uploading a file with raw data, we encrypt the file and upload the encrypted file to IPFS to secure the privacy of the file's data.

Regarding file download, the application asks for the user's identity information. Specifically, the application prompts the user for their email address just for experimental purposes. The application receives the email address the user enters and sends it to a smart contract on the local blockchain. The smart contract runs a method to verify if the email address is valid and trusted or not. On successful verification, we call the Get API from the IPFS client to get the file to the user. On failed verification, the application shows an error message to the user and will not download the file from IPFS.

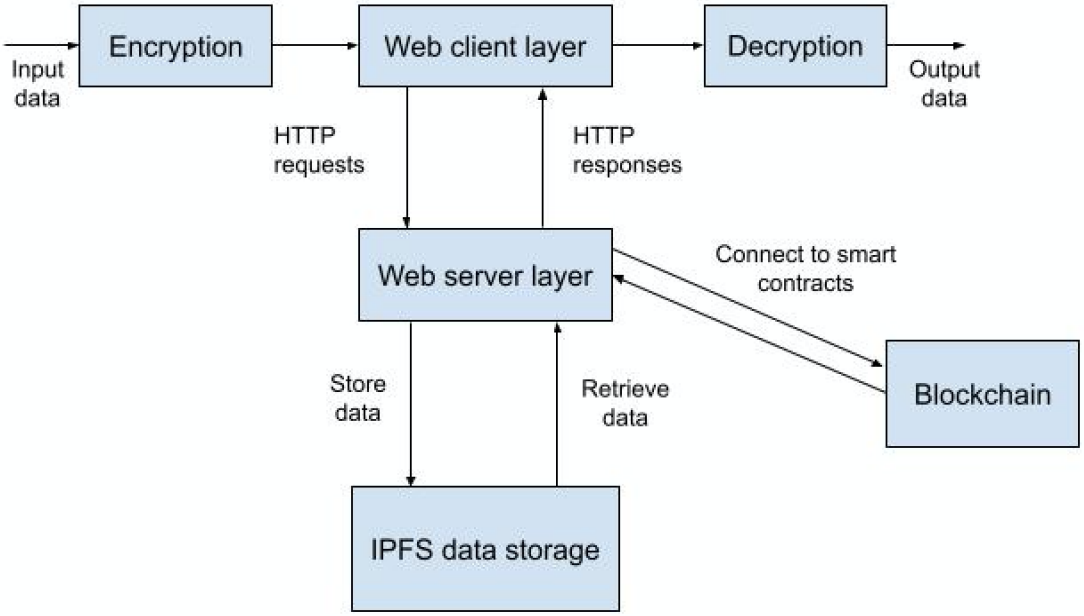

Figure 5. High level architecture diagram

At a high level view, the top layer of the application is the web client layer. This is the user interface that runs on a browser, so the user can interact with and perform operations in the application. The web client layer talks to the web server layer through the traditional HTTP protocol. The web server consists of all the implementations to support file upload and download, interact with smart contracts on the blockchain, and communicate with the IPFS data storage layer. The data storage layer is the bottom layer that stores and manages the data. The web server talks to the data storage layer through proper API calls to store new data or retrieve existing data.

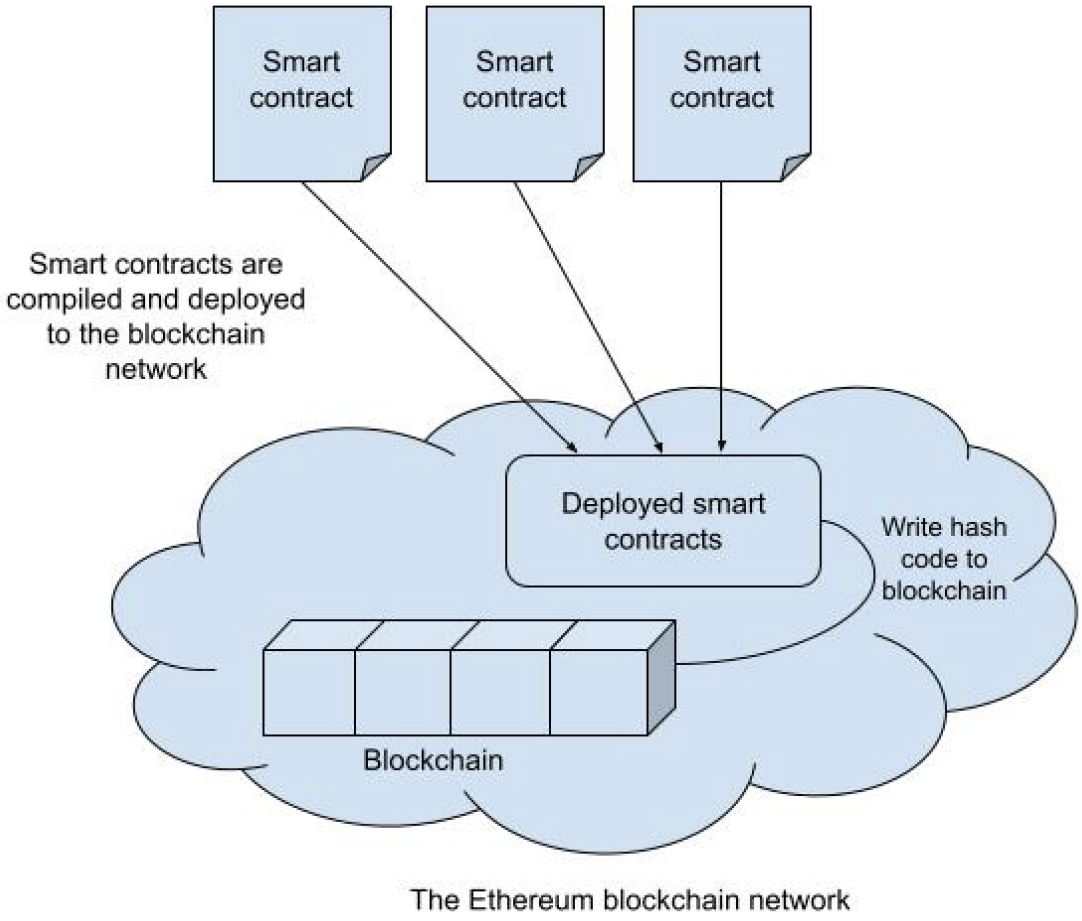

Figure 6. Smart contracts in the Ethereum blockchain

The blockchain network is home to deployed executable smart contracts. When the web server needs to interact with a smart contract, it uses the contract's network address to connect and then call a method in the smart contract. The contract executes the method to perform an operation requested by the application, such as writing the hash code to the blockchain or verifying the user's identity. In the case of uploading new data, the smart contract will write the hash code to the blockchain by adding a new block that contains the hash and links to the previous node on the blockchain. The contracts are deployed to and running on the blockchain network, as illustrated in Figure 6 above.

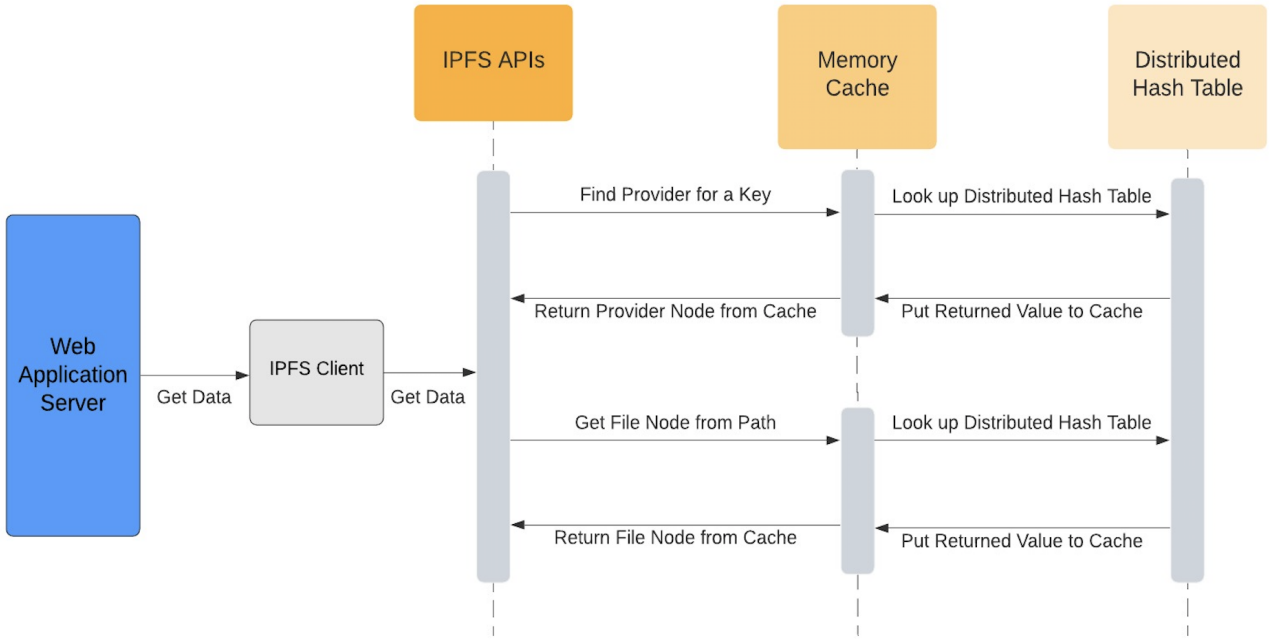

Figure 7. Sequence diagram of data retrieval in IPFS

Figure 7 shows the sequence diagram of data retrieval in IPFS: The web application server uses an IPFS client to talk to a remote IPFS network of nodes or the local IPFS daemon. In our experiments, we run the IPFS daemon locally, so we can test the code changes we make to improve the performance of IPFS data retrieval. We add key-value memory cache to two APIs in

IPFS. One API is to get the root file node from a file hash, and the other API is to find provider nodes that can provide the corresponding data from a key. A key is the content addressed identifier of a file. Besides querying data, the web application also utilizes the IPFS client's Add API to add new data to IPFS.

IV. IMPLEMENTATION

4.1 Implementation Approach

As we are trying to improve the security of data stored in IPFS and the performance of data retrieval from IPFS, we have decided to use encryption/decryption, access control using smart contracts, migration of the web application from localhost to a remote IPFS network, and memory cache. Below are the snapshots of two pages of the application for file upload and file download.

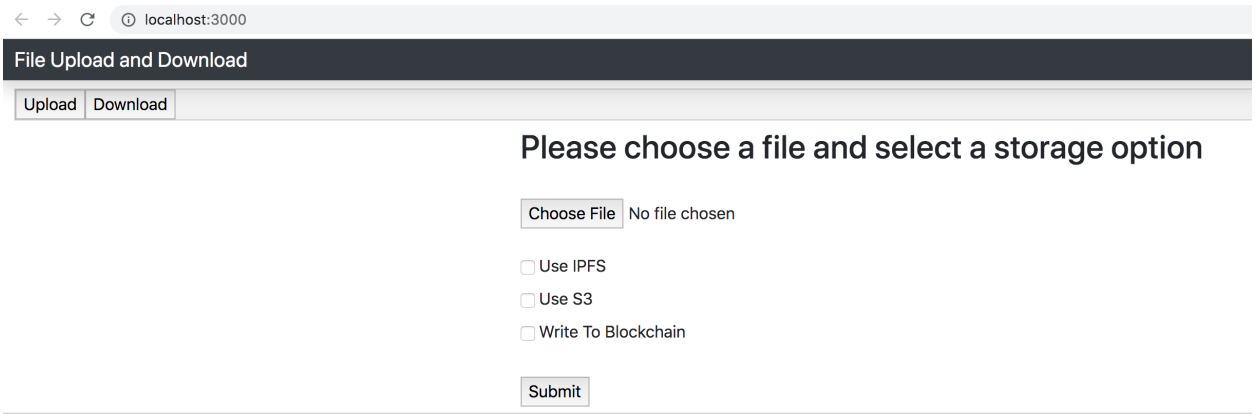

Figure 8. Upload entry page of the application

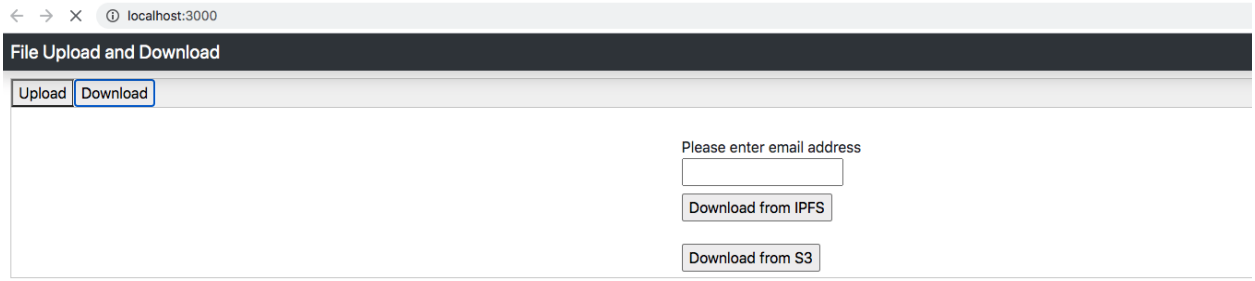

Figure 9. Download entry page of the application

In terms of security, we do not want the user's actual data to float around in a distributed IPFS file system since the data can be stored on multiple servers, which increases the risk of the data getting hacked or stolen by a malicious party. Although IPFS hashes the file content, anyone who has access to the hash can retrieve data from IPFS. It creates a security gap there. To improve the security, we are using asymmetric encryption to encrypt the data before it is uploaded to IPFS.

Smart contracts offer a way for us to implement access control over the data on IPFS. We can implement logic in a smart contract to verify certain things, such as the username, email address, etc. from a user when they try to download a file from IPFS. This will prevent an unknown user from accessing the data because they do not have all the right information of the known user.

Migrating an application from the localhost to remote IPFS helps improve the availability and accessibility of the application. We did this experiment to prove that once the application is hosted on IPFS, it can be accessed from any computer, regardless of whether our localhost machine is on or off.

To improve the performance of data retrieval from IPFS, we use a key-value memory cache to cache the responses of some major internal APIs or method calls in IPFS. The key in the cache will be the content identifier because it is unique for the same content. If the cache contains the key, we can immediately return the value without doing any extra work, so it helps make data lookup on IPFS faster.

4.2 Encryption and Decryption

Before getting uploaded to IPFS, a file is encrypted with the public key of the recipient. When the recipient downloads the file using the hash from IPFS, they can use their private key to decrypt the file to get the actual data. We used asymmetric encryption to encrypt a text file by the

public key of the recipient, so that only the right recipient has the associated private key to decrypt the file. If someone else has the hash and tries to download the file, they cannot decrypt and see the file content because they lack the correct private key. It improves the security of data on IPFS since only the intended recipient has the right private key to be able to decrypt the downloaded file.

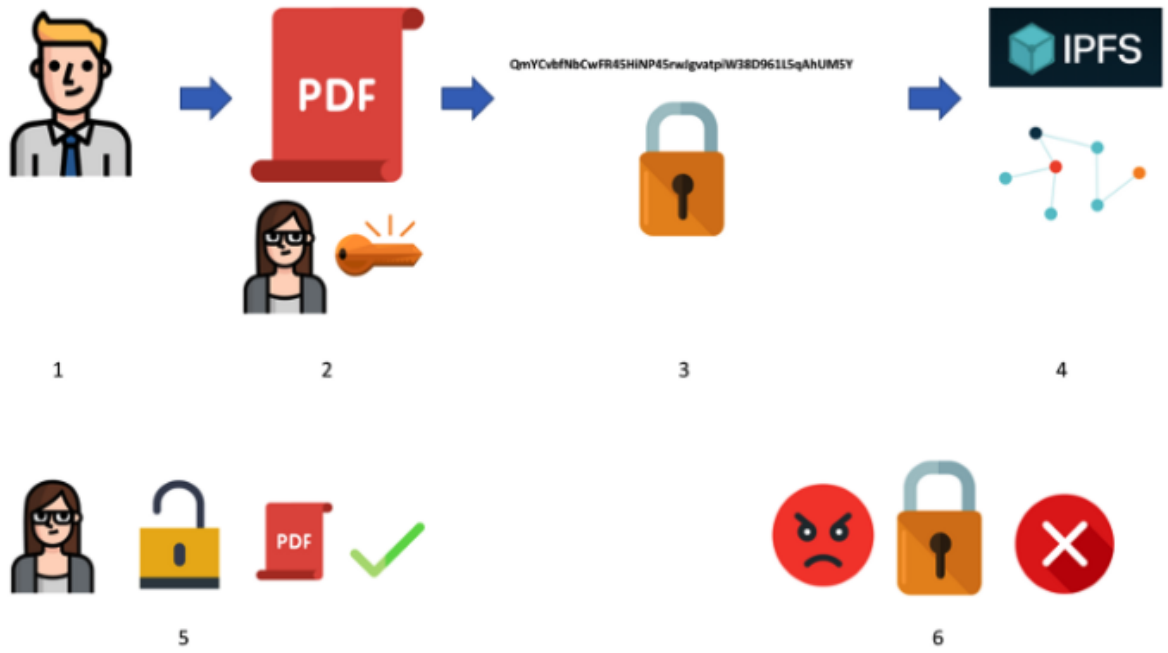

Figure 10. Asymmetric encryption when uploading to and downloading from IPFS

After the data has been saved in IPFS, IPFS returns the hash of the file content. We contact the proper smart contract to write this hash to the local blockchain, so that no one can tamper the hash thanks to the blockchain's immutability property. The hash will then be copied and distributed to other peer nodes, which makes it hard to be lost since multiple copies of the hash are on the blockchain.

```
pragma solidity 0.5.16;

contract File {
  string fileHash;

  // Write function
  function set(string memory _fileHash) public {
    fileHash = _fileHash;
  }

  // Read function
  function get() public view returns (string memory) {
    return fileHash;
  }
}
```

Figure 11. Code snippet on a smart contract to write/read the file hash to/from the blockchain

4.3 Access Control using Smart Contracts

Smart contracts have been proven useful and effective in Ethereum blockchain based systems to ensure a transaction between two parties is handled correctly as per the mutual contract. Smart contracts are Turing complete, so they can be used for various purposes. Ethereum blockchain is stateful, so it can record changes executed by smart contracts. Additionally, the blockchain is appended only and immutable, and any modification to the blockchain is agreed upon through consensus of participating nodes. Therefore, it eliminates chances of a malicious user trying to tamper with the data on the blockchain. All of the above enables us to implement access control as well as adding, modifying, and removing conditions for accessing data.

We have integrated smart contracts to our application to implement the access control of the data on IPFS. For our experiment, we have written the logic in a smart contract to check the user's email address. The smart contract is compiled and deployed to our local blockchain. When a user tries to download a file from IPFS, they are prompted to enter their email address. This

email address is sent to the application server, and the server calls a method on the smart contract for email verification. If the verification succeeds, the server proceeds with fetching the requested data from IPFS. Otherwise, we show an error message to the user that they do not have access to the requested data.

```
pragma solidity 0.5.16;

contract AccessControl {
  string email;

  constructor () public {
  }

  // Write function
  function set(string memory _email) public {
    email = _email;
  }

  // Read function
  function get() public view returns (string memory) {
    return email;
  }

  function checkEmail() public returns (bool) {
    if(bytes(email).length != bytes("vule@sjsu.edu").length) {
        return false;
    } else {
        return keccak256(bytes(email)) == keccak256(bytes("vule@sjsu.edu"));
    }
  }
}
```

Figure 12. Code snippet on a smart contract to verify an email address

```
const accessControl = new window.web3.eth.Contract(AccessControl.abi, networkData.address)
const accounts = await window.web3.eth.getAccounts()

accessControl.methods.set(email).send({from: accounts[0]})
  .then((r) => {
    this.setState({ email })
    console.log("Set email to smart contract")
  }).catch((err) => {
    console.log("Failed to set email, err " + err.message)
  })

var success = await accessControl.methods.checkEmail().call()
```

Figure 13. Code snippet to call the smart contract from the application server

4.4 Migrate Web Application from Localhost to Remote IPFS

Speaking of hosting a web application, we usually think about using a dedicated centralized hosting service, such as Google, Amazon, then register a domain name, and set up a DNS record to point the domain to the hosting server. Although these services can provide quite reliable access and availability to our application, what happens if the centralized service goes down, or the domain name service is not available? Our web application will not be accessible any more.

IPFS provides a great way to decentralize the hosting of an application where multiple nodes can host the application, so the application can be accessed in case one or a few nodes go down. Whereas the centralized server provides access to a website through an HTTP protocol, an application running on IPFS will be accessed through the IPFS peer to peer protocol or the IPFS Gateway API. It is even better that once the application is on IPFS, we can pin it to the local node using the hash output from IPFS in order to make sure the application stays on the IPFS network [8].

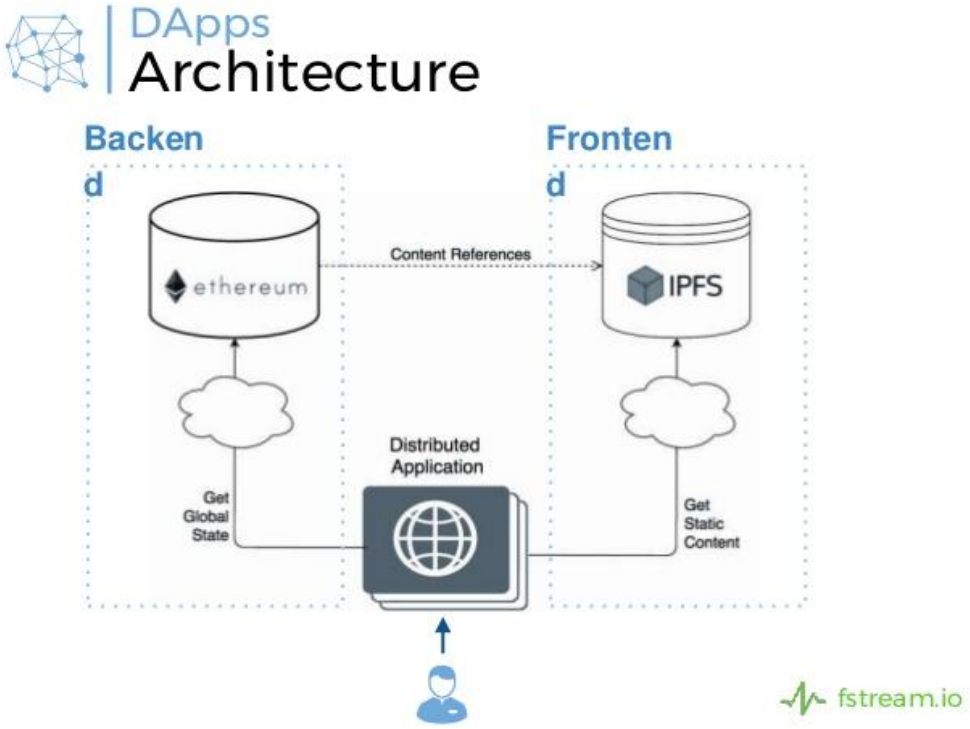

Figure 14. Distributed app architecture supported by IPFS and Ethereum blockchain

We have used a tool on fleek.co to deploy our application to a distributed IPFS network. Once deployed, we create a domain name on GoDaddy and change the CNAME record in our domain's DNS to have a better URL to access the application.

4.5 Key-value Memory Cache

We have used a third party caching library to add caching to the IPFS source code. The memory cache is useful when a request for the same key is sent to a server multiple times within a certain amount of time. The first time the request will go through the normal flow to retrieve everything it needs to retrieve the requested data. The subsequent requests for the same key will hit the cache, so we can avoid unnecessary calls to other methods. The cache is built upon a thread safe map with the key of type string, and the value of type interface, so that the value can store objects of different types. The cache can be accessed by multiple methods as well.

There are three main operations on the cache.

- Cache set: used to set a key-value mapping in the cache. If the key already exists in the cache and has not expired, the key-value pair will be replaced with the new value.
- Cache get: used to get the corresponding value of a key from the cache. If the key does not exist in the cache, it will return null. This operation returns a pair of the associated value and a boolean indicating if the value was found or not.
- Cache evict: used to delete a key-value entry in the cache when the key already expired.

The cache's methods synchronize access to the cache's internal data mapping to avoid inconsistent reads and data integrity issues. The cache constructor takes an expiration duration and a cleanup interval. Entries in the cache will expire after the expiration duration. A cleanup routine runs periodically at the specified interval to delete expired entries. It ensures the cache only stores recently accessed data. It is possible to make the cache entries never expire by setting the expiration time to -1. The cache will maintain the entries forever, so it will require manual deletions to clean up the cache.

4.6 Methods

4.6.1 getFileNodeFromPath()

This method takes an IPFS hash path to a file and resolves to a file root node that points to the start of the file. The method queries the directed acyclic graph (DAG) which is the underlying data structure of IPFS and returns a node pointer to the file. Since the hash path is based on the file content and is the same for the same content, we can put the hash path and the resolved file node in the cache. Future requests with the same hash path will hit the cache and immediately return the file node.

This method first resolves the given path to the last valid resolvable path. It uses the content identifier in the last resolvable path to query the DAG. The DAG will return a root node that contains links to other nodes in the graph representing the requested data. When receiving the root node, the method creates a new Unix file system based file node. This node can represent different hierarchies of the file system, such as a file, a directory, or a symbolic link to a file.

4.6.2 findProvidersAsync()

While getFileNodeFromPath() returns the file root node to a specific given hash path, this findProvidersAsync() method searches for peers that are close to the current node and can provide a given key. The method takes a content addressed identifier and a count, then returns a set of peers that have or know where to get the requested content. A content addressed identifier is formed by a version, a content type, and the hash of the content. The count specifies how many peers we want to get. If the count is zero, the method returns an unbounded number of peers. Because the content identifier is unique for the same content, we cache the content identifier and the set of peer providers. Therefore, future calls to this method with the same identifier will be able to directly get the providers from the cache.

This method internally tries to find providers from the local network as well as the wide area network. In each network, it creates a channel which is responsible for sending back a peer's address when it is found. Since a peer can be found again and again, the method keeps a local map that stores a newly found peer. We just ignore the peer that has already been found and is in the map. If at the end of the method, we cannot find any new peers, we publish a query event to the query event channel. The query event has the context of the initial data request.

4.7 How Everything Works Together

Now that we have finished all the necessary parts to improve the security and performance of our web application. We have run some simple tests on the same application workflow to illustrate how everything works together.

We first encrypt a text file using the recipient's public key and upload it to IPFS. We use a file large enough, so that during the upload, the file is broken into multiple data blocks. The blocks are processed and can be stored on different nodes on IPFS. The upload request returns the hash of the file content which we can use to retrieve the file later.

When the file is successfully uploaded, we try to download the file from IPFS. The application requires the user to enter their email address before they can download. We send a download request along with the user's email address to the application server. The server calls the smart contract that has already been deployed to the local blockchain to verify if the entered email address is correct and trusted. If the verification fails, the application pops up a message to inform the user that they are not authorized to download the file.

After the email has been verified, we proceed to call the IPFS Get API to retrieve the file's data. The API takes the hash we received from the upload and tries to find data providers that have the file and are ready to return the file. During the process of finding data providers, the two methods findProvidersAsync() and getFileNodeFromPath() are internally called. When each method receives the needed response objects, it puts the response objects into the memory cache. When the Get API returns, from our web application side, we should receive the file's content in the response.

Now we try to download the same file again and measure the time it takes in both methods findProvidersAsync() and getFileNodeFromPath(). We should observe that each method is taking much less time than before thanks to the memory cache.

Since the memory cache is refreshed periodically and invalidates data entries that have not been accessed for some expiration time, we wait long enough for the data to be invalidated and try to download the same file again. We again measure the time elapsed in both the aforementioned methods. We should see that both methods are taking a much longer time because the requested data is now missing in the cache, so the methods have to go through a full normal flow to retrieve the data.

V. EXPERIMENTS AND RESULTS

The purpose of the project is to improve the security and performance of web applications when using the distributed file system IPFS. We integrated various methods into our application to make it more secure and faster. The methods we used are encryption/decryption, access control using smart contracts, and memory cache. We have done 4 experiments where 3 of them demonstrate the result of each improvement method we have implemented, and 1 experiment shows the application is completely running on the remote IPFS.

Table 2: Summary of the experiments and their results

Experiment	Result
Encryption and decryption	Only the expected recipient can decrypt the file after downloading from IPFS because they have the correct private key.
Access control using smart contracts	Only users with trusted and valid identity can access the data on IPFS
Migration of the application to IPFS	The application can always run and be accessible even when the local server shuts down.
Memory cache	Improve the latency to find node providers for a key as well as finding the data's root node from an IPFS hash path. The latency goes down from more than 300 ms to about 10 ms for these operations.

5.1 Encryption and Decryption

For the first experiment, we used encryption and decryption to protect the privacy of the data we upload to IPFS. We first generated a pair of a public key and a private key using OpenSSL commands.

We ran the following commands to encrypt the file test.txt, output the encrypted file encrypted.txt, and uploaded the encrypted file to IPFS.

- openssl rsautl -encrypt -pubin -inkey public.key -in test.txt -out encrypted.txt
- ipfs add encrypted.txt

The below screenshot shows the contents of the original file and the encrypted file.

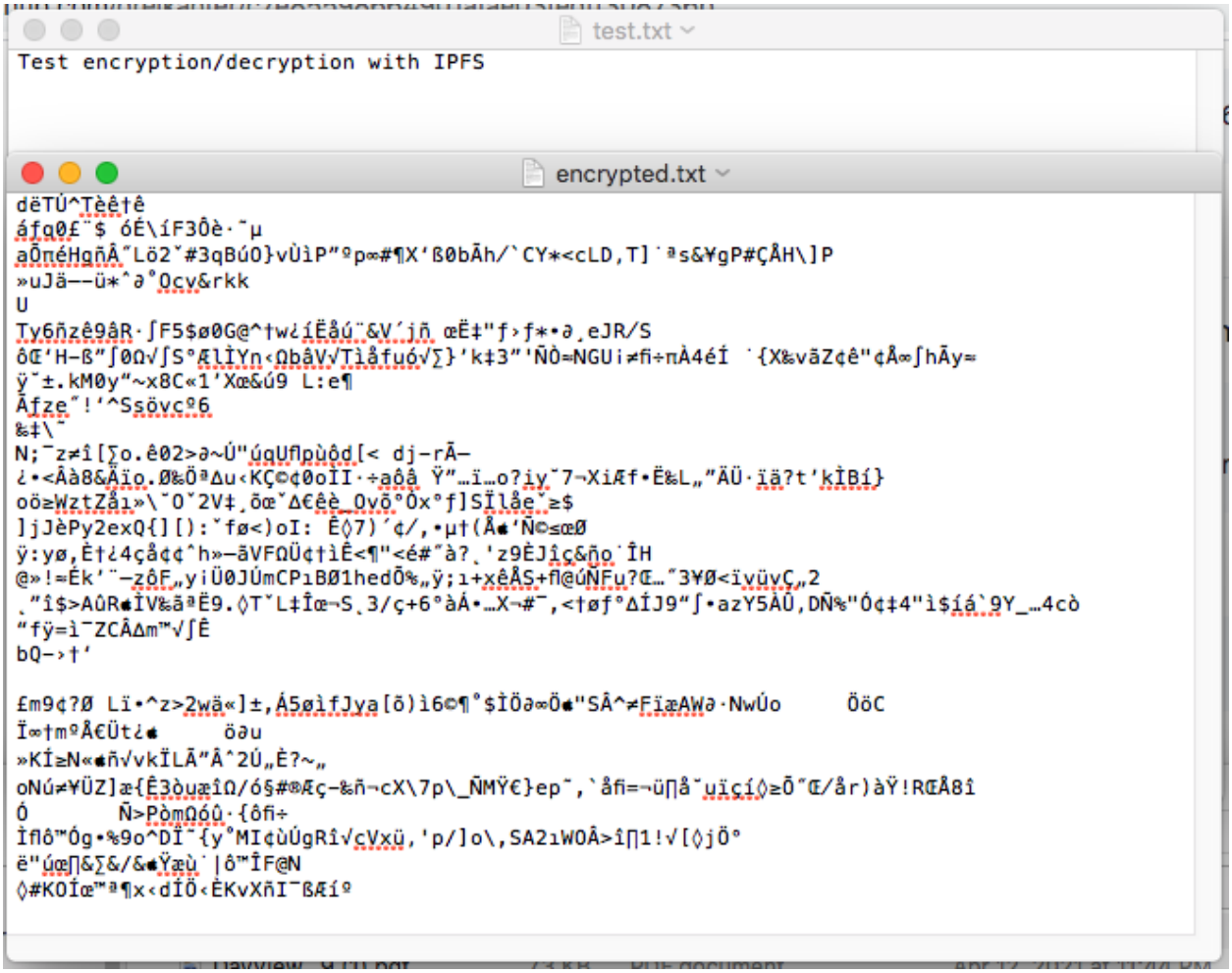

Figure 15. The contents of the original file and the encrypted file

The *ipfs add* command returned the hash of the file as "QmWnGk5okhJocwLRS8QDkMnmzpiGHXdu6UrrqA3a88XqXX". We used this hash to

download the file back from IPFS and then decrypted the downloaded file. We ran the following commands.

- ipfs get QmWnGk5okhJocwLRS8QDkMnmzpiGHXdu6UrrqA3a88XqXX
- openssl rsautl -decrypt -inkey private.key -in QmWnGk5okhJocwLRS8QDkMnmzpiGHXdu6UrrqA3a88XqXX -out decrypted.txt

The downloaded file was saved with the same name as the hash "QmWnGk5okhJocwLRS8QDkMnmzpiGHXdu6UrrqA3a88XqXX". We decrypted this file to create an output file called decrypted.txt

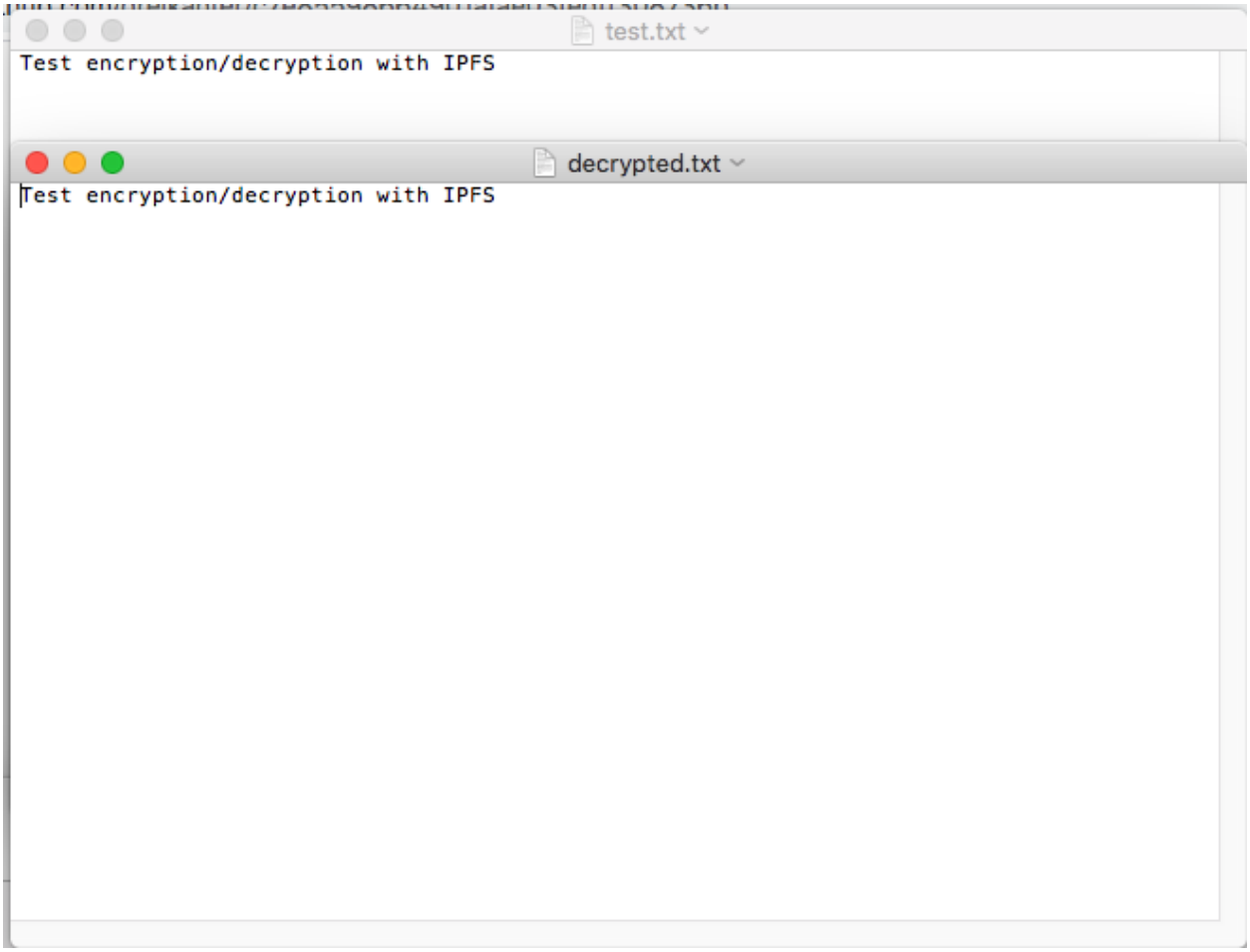

Figure 16. The original file and the decrypted file have the same content

5.2 Access Control using Smart Contracts

In the second experiment, we used smart contract based access control to verify the user's identity before giving them permission to download a file on IPFS. When we downloaded a file from IPFS, we were required to enter an email address for the smart contract to validate. For experiment purposes, we hard coded this email address vule@sjsu.edu in the smart contract, and the email verification would fail for any email different from the hard coded email. When the entered email address passed the verification, we showed a success message to the user and started downloading the file from IPFS.

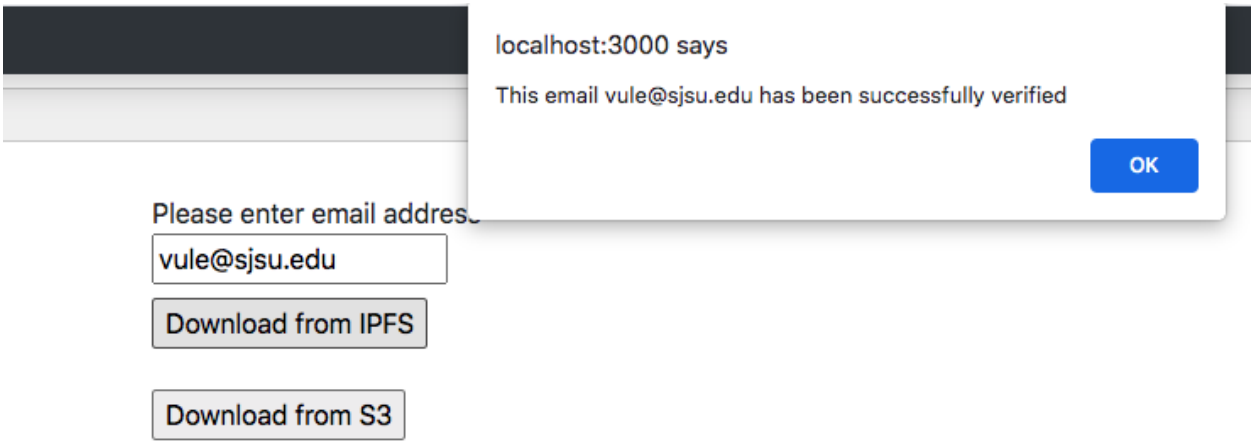

Figure 17. The success message for email address verification

When we put in an unexpected email address, it would fail the verification by the smart contract. We showed a failure message to the user and did not allow them to download the file.

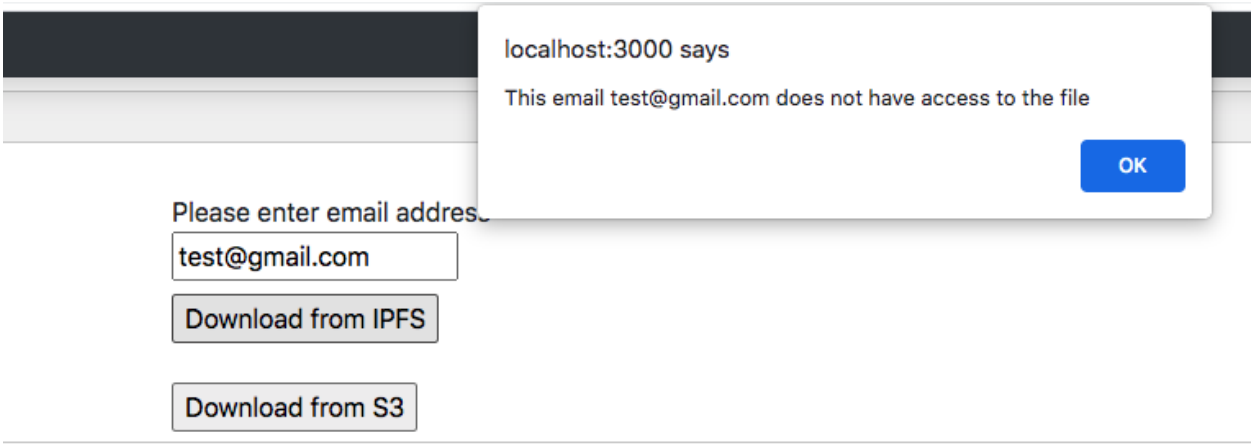

Figure 18. The failure message for email address verification

5.3 Migration of the Web Application to IPFS

We have migrated the web application from the localhost to a remote IPFS network, so the web application can be accessed even though the local server is not running. We also set up an SSL connection and a custom URL https://sjsu-cs298.on.fleek.co for secured and easy access to the application.

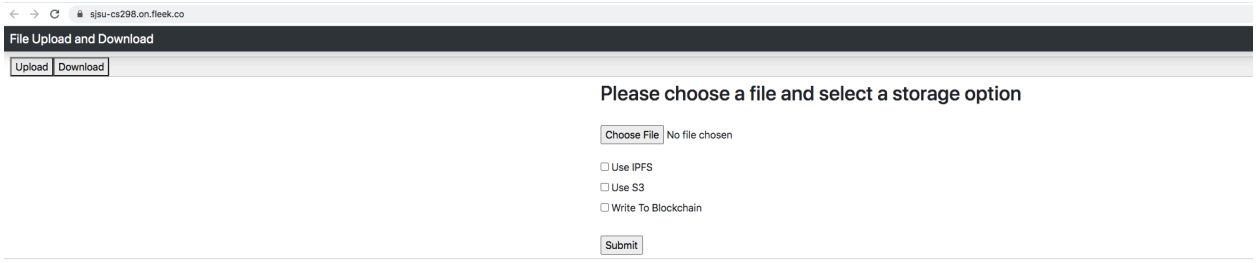

Figure 19. Web application running on remote IPFS (upload page)

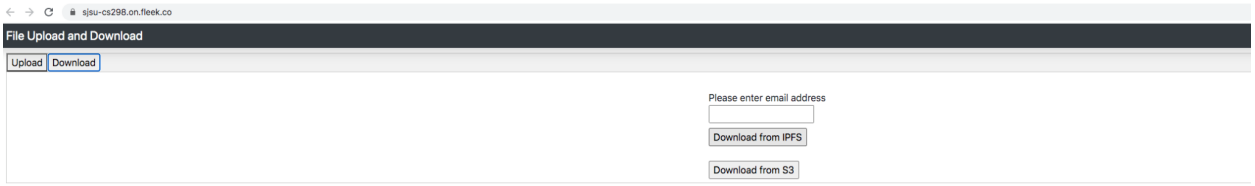

Figure 20. Web application running on remote IPFS (download page)

5.4 Memory Cache

We experimented uploading and downloading files of different sizes and measured the time it took to get the root file node with and without the cache. For each file, we ran the same experiment 10 times and took the average latency. The results are shown in Figure 21.

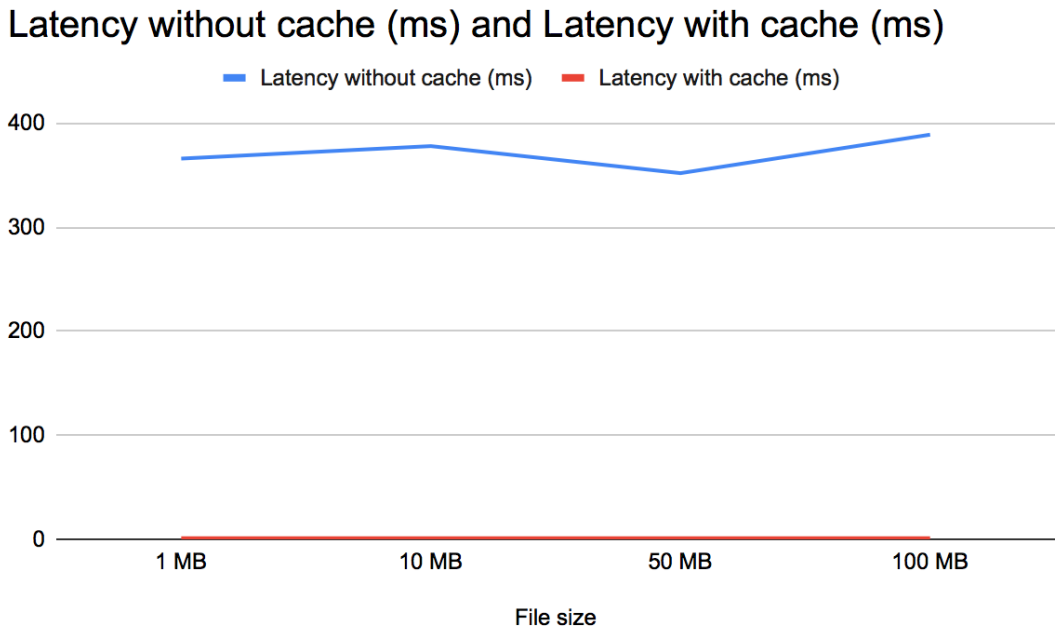

Figure 21. Comparison of the latency in getting the root file node with and without the cache

We can see that without the cache, the average latency fluctuates from 300 ms to 400 ms. In our experiment, sometimes it can take up to a few seconds to get the root file node because it depends on when the lookup algorithm finds the node.

We also measure the download latency for different file sizes, shown in Figure 22. With the

cache, it only takes about 10 ms to get the root file node, and the latency is pretty consistent. The latency that the user observes when downloading a file from the application will be higher due to the Internet latency to transfer the data. The Internet latency is insignificant and not noticeable for a 1Mb file, and it is increasing as the size of a file gets bigger.

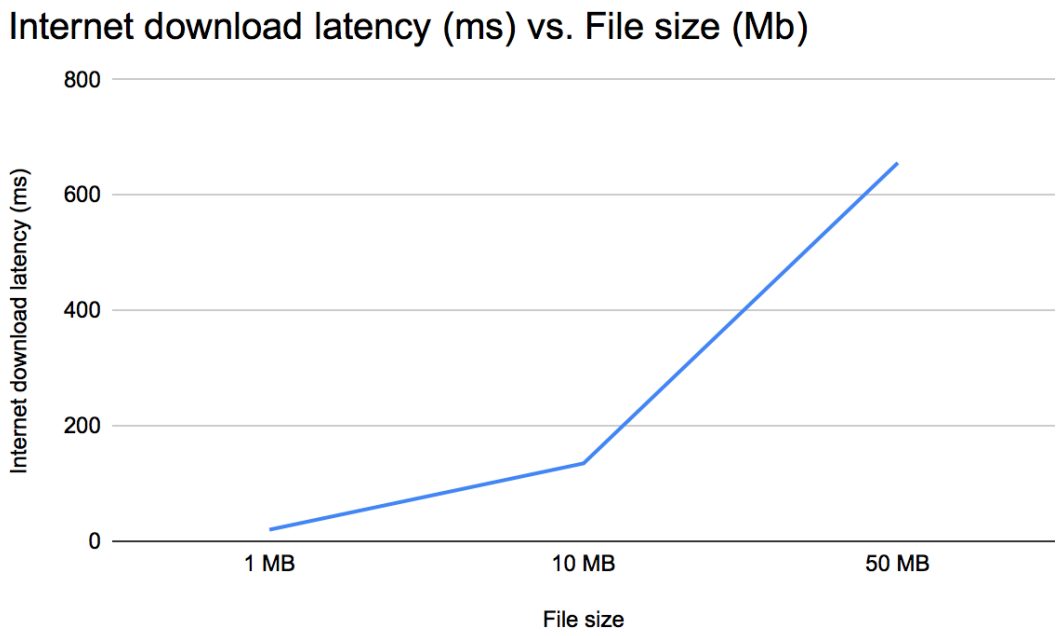

Figure 22. Internet download latency for different file sizes

We also tried varying the expiration time of the memory cache; the results are shown in Figure 23. We pre-uploaded 100 different text files of about 1 Mb each to IPFS. We tried setting the expiration time to 10 seconds, 30 seconds, 1 minute, and 5 minutes, and for each expiration time, we ran a script to continuously download random files from the pre-uploaded files on IPFS and measured the latency. The resulting latency is drawn in the chart below. As the chart illustrates, when the expiration time increases, the latency decreases because more entries are kept in the cache, and we get more cache hits in the download operation.

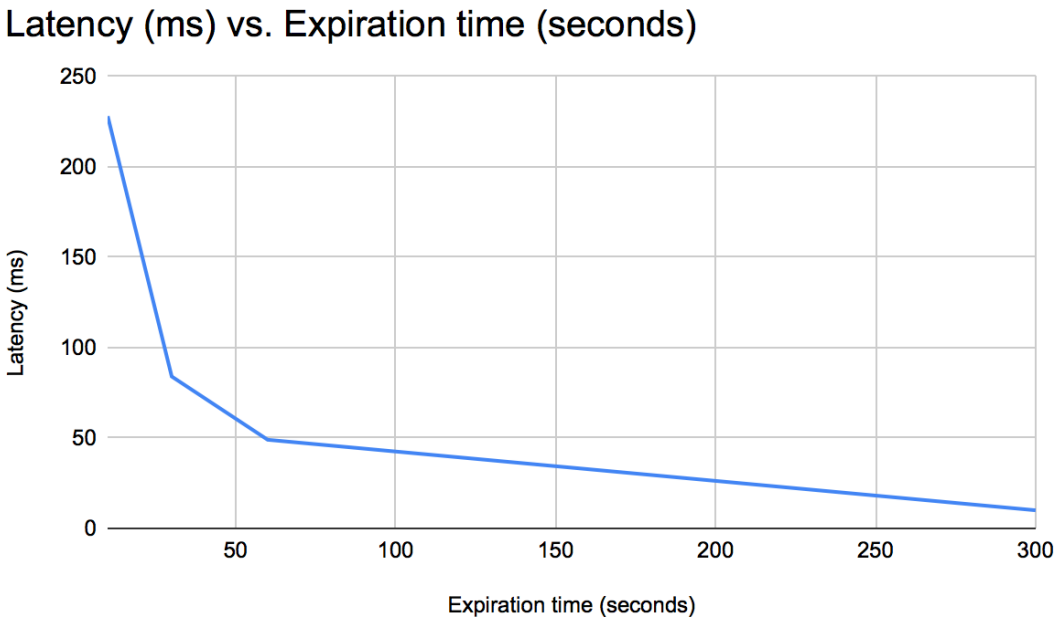

Figure 23. Latency vs. expiration time of the memory cache

The memory cache is backed by a key-value map in Go language. Since a map can have a dynamic size, when more data is added to the map, the map will get more slots internally. Therefore, the maximum amount of data in the cache will depend on the available heap size and the operating system where IPFS is running. For example, a map in a 32-bit operating system can have 2^{31} - 1 slots, and a map in a 64-bit operating system can have 2^{63} - 1 slots. Because of the above reason, initializing a map with a different size for our memory cache will not result in any difference in terms of the latency to get the data. However, the size of the cache is limited by the heap size and the memory available in the operating system.

Note that there is a possible disadvantage of using the memory cache. When the data requested from IPFS is too large that the cache cannot hold, it will result in a cache miss. When many cache misses are happening, it will slow down the performance of IPFS. Depending on the application's use cases, if IPFS with the integrated memory cache is not a good fit, we can consider other alternative distributed file systems, such as Ceph, GlusterFS, and HDFS. HDFS

stands for Hadoop distributed file system which can store, process large unstructured data, and provide high throughput access. Ceph is a POSIX-compliant network file system that offers high performance, large data storage, and is highly compatible with legacy applications. GlusterFS is also a fast and scalable distributed file system which has caching of data and metadata. GlusterFS can be integrated with HDFS to process large data for data analysis purposes [6].

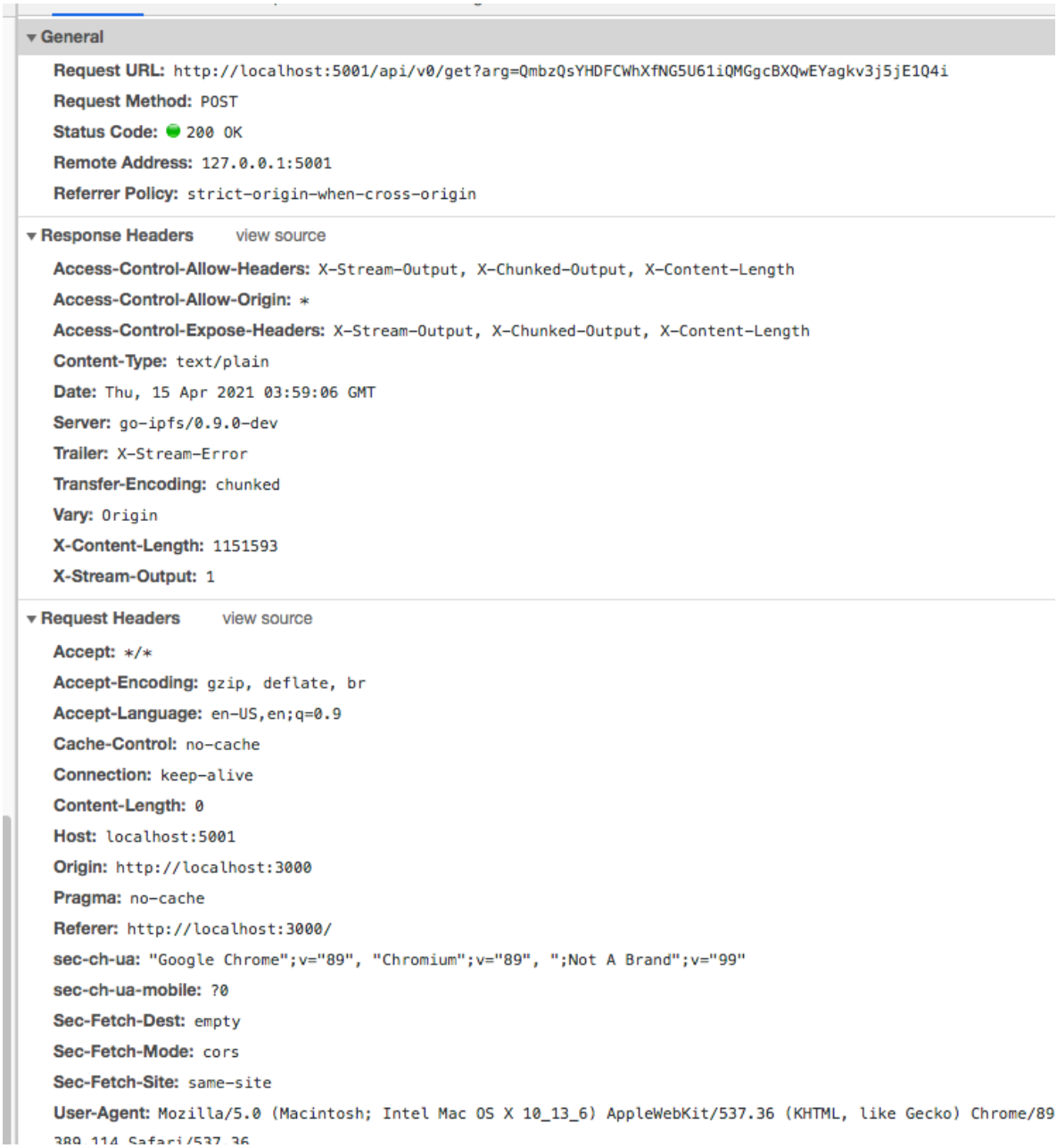

Figure 24. The request headers and parameter for Get API call to IPFS

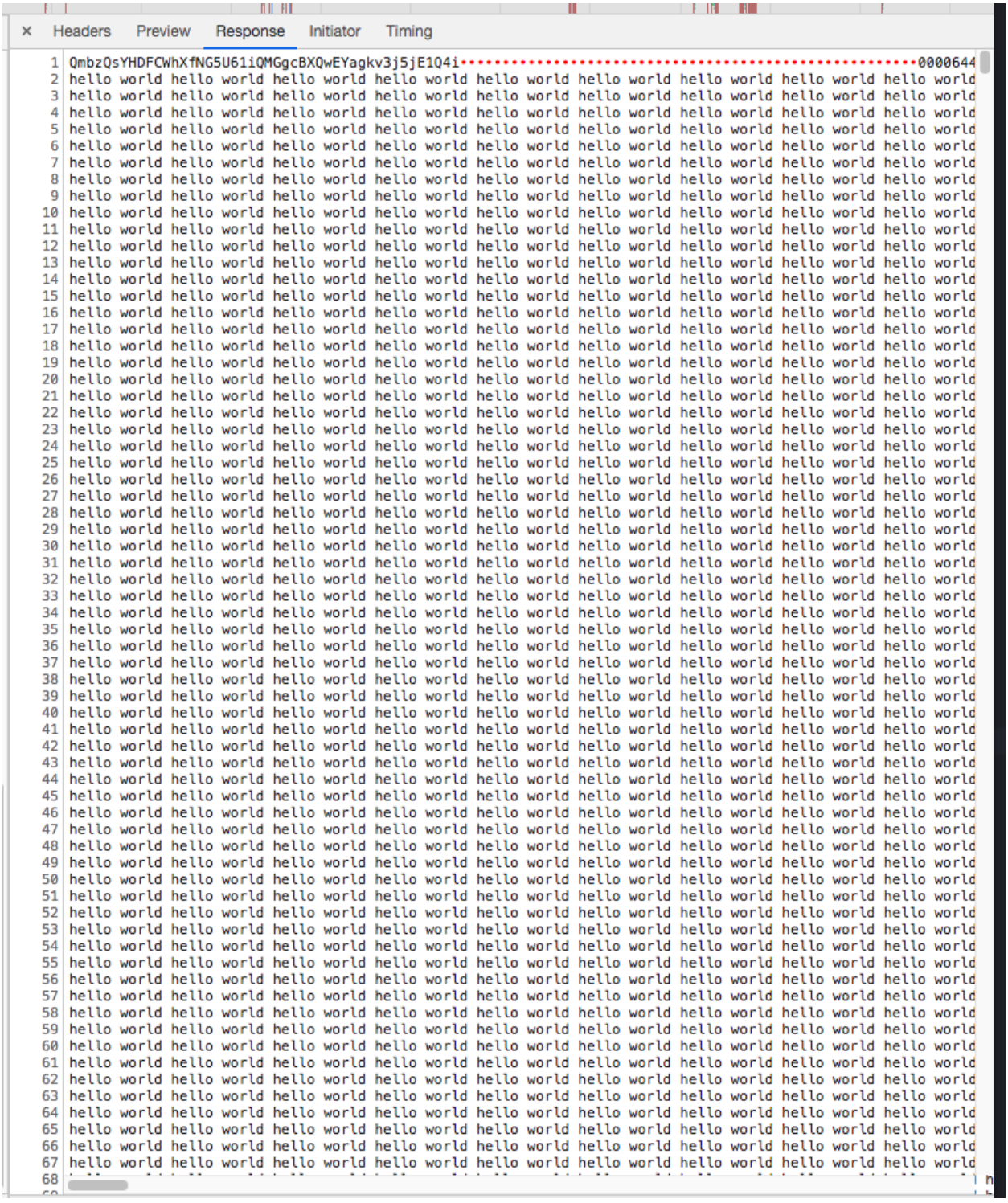

Figure 25. The response shows the content of the downloaded file from IPFS

5.5 Tradeoff Between Performance and Security

There is a computing cost for each added security mechanism in the appli- cation, so it incurs a tradeoff between performance and security. When an additional layer of security is integrated, the application's server has to do more computing work in the security check, so the application can run a lit- tle slower. If security is more important, we need to sacrifice performance of the application, and vice versa. Therefore, it is important to choose the right balance between performance and security to meet the requirements and use cases of the application.

We have measured the performance impact in our application for each security method. We upload and download files of 1 Mb in size. The encryption and decryption using OpenSSL each takes about 50 ms. The call to the smart contract to verify the user's identity takes between 20 to 70 ms. In short, the security implementation adds an additional 50 ms to the file upload flow and an extra 70 ms to 120 ms when downloading a file.

As we know from the experiments, IPFS with the memory cache only takes about 10 ms to resolve the root file node, so the additional latency from the security protection is about 7 to 12 times longer on average for a file download. From the user experience, the added latency of about 100 ms in a download should not be noticeable. However, if we want to upload a very large file, we may need to split it into chunks and encrypt each chunk at a time because there is a size limit in OpenSSL encryption. This splitting and encrypting of a large file may add a significantly longer latency for uploading a file, which is something we need to consider.

VI. CONCLUSION AND FUTURE WORK

Our research and experiments show that it is imperative to protect the security and privacy of data on a distributed network, and we have used several methods in our application to improve it. Data encryption ensures the data can only be seen by the authorized user who has the right key to decrypt the data. Access control by smart contracts provides another layer of security check to verify the user's identity before giving the user permission to access the data. Combining the two methods can significantly improve the security and lower the risk of data being stolen by malicious users. Some future work to be done in this area will be to add more access control policies to smart contracts to verify the user's identity, such as verify the user's public key and signature. We can also have a blockchain based service that encrypts and manages the user's secret key, and an application which wants to access the data can call the service to get the key and decrypt it. This key management service can use smart contracts' access control to authenticate the user before sending them the key [18, 19].

Besides the security concern, our experiments demonstrate performance improvement in IPFS by using key-value memory caching. The cache helps significantly reduce the time it takes to find a data provider or the root node of an underlying directed acyclic graph in IPFS, which in turn makes it faster to retrieve data from IPFS. In the future, another way we can try to add on top of caching is to use recursive lookup instead of the current iterative lookup. When the iterative lookup asks a node for the associated value of a key, if the node does not have the value, it will return a list of other nodes closer to the requested value. The same process continues until it finds a node that actually has the value and can return it. With the recursive lookup, when we ask a node for a value, and the node does not have the value, it will ask another closer node which will ask another closer node. Once the value is found, it will be returned to the original

requestor following the same chain of requests in the reversed order. The recursive lookup can reduce a lot of network traffic between nodes and the latency in processing requests and responses in each node, so it will certainly improve the performance of data retrieval in IPFS.

The performance can degrade when extremely large data files are trans- ferred because the entire data will not fit in the memory cache. We will more likely experience cache misses. Another future work is to find a solution to han- dle this situation and try to achieve the optimized performance of the cache as well as IPFS when requesting large data from IPFS.

VII. REFERENCES

[1] Shermin V (2019) Blockchains & Distributed Ledger Technologies. BlockchainHub. Accessed on: Apr 12, 2020. [Online]. https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/

[2] Dragonchain (2019) What Different Types of Blockchains are There? Accessed on: Apr 11, 2020. [Online]. https://dragonchain.com/blog/differences-between-public-private-blockchains/

[3] Durcevic S (2019) Cloud computing risks, challenges & problems businesses are facing. Datapine.com. Accessed on: Apr 17, 2021. [Online]. https://www.datapine.com/blog/cloud-computing-risks-and-challenges/

[4] Hassanzadeh-Nazarabadi Y, Kupcu A, Ozkasap O (2019) LightChain: A DHT-based Blockchain for Resource Constrained Environments. ArXiv abs/1904.00375

[5] Hu Y, Manzoor A, Ekparinya P, Liyanage M, Thilakarathna K, Jourjon G, Seneviratne A (2019) A Delay-Tolerant Payment Scheme Based on the Ethereum Blockchain. IEEE Access 7(6):33159–33172, DOI 10.1109/access. 2019.2903271, URL https://dx.doi.org/10.1109/access.2019.2903271

[6] John BK (2020) Ceph vs. GlusterFS vs. MooseFS vs. HDFS vs. DRBD. ComputingForGeeks. Accessed on: Jun 4, 2021. [Online]. https://www.datapine.com/blog/cloud-computing-risks-and-challenges/

[7] Khudhur N, Fujita S (2019) Siva - The IPFS Search Engine. 2019 Seventh International Symposium on Computing and Networking (CANDAR), Nagasaki, Japan, 2019 pp. 150-156. DOI 10.1109/CANDAR.2019.00026, URL https://doi.ieeecomputersociety.org/10.1109/CANDAR.2019.00026

[8] Kohorst L (2020) Decentralizing your Website. Medium. Accessed on: Mar 27, 2021. [Online]. https://towardsdatascience.com/decentralizing-your-website-f5bca765f9ed

[9] Lipton A, Levi S (2018) An Introduction to Smart Contracts and Their Potential and Inherent Limitations. The Harvard Law School Forum on Corporate Governance. Accessed on: Mar 27, 2021. [Online]

https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/

[10] Nakamura Y, Zhang Y, Sasabe M, Kasahara S (2020) Exploiting Smart Contracts for Capability-Based Access Control in the Internet of Things. Sensors 20(6):1793–1793, DOI 10.3390/s20061793, URL https://dx.doi.org/ 10.3390/s20061793

[11] Naz M, Al-Zahrani FA, Khalid R, Javaid N, Qamar AM, Afzai MK (2019) A Secure Data Sharing Platform Using Blockchain and Interplanetary File System. Sustainability. 11. 10.3390/su11247054

[12] Rahulamathavan Y, Phan RC, Rajarajan M, Misra S, Kondoz A (2017) Privacy-preserving blockchain based IoT ecosystem using attribute-based encryption. Proc IEEE Int Conf Adv Netw Telecommun Syst pp 1–6

[13] Salman T, Zolanvari M, Erbad A, Jain R, Samaka M (2019) Security Services Using Blockchains: A State of the Art Survey. IEEE Communications Surveys & Tutorials 21(1):858–880, DOI 10.1109/comst.2018.2863956, URL https://dx.doi.org/10.1109/comst.2018.2863956

[14] Shen J, Li Y, Zhou Y, Wang X (2019) Understanding I/O Performance of IPFS Storage: A Client's Perspective. IEEE Access pp 24–24

[15] Song L, Li M, Zhu Z, Yuan P, He Y (2019) Attribute-Based Access Control Using Smart Contracts for the Internet of Things. ScienceDirect pp 2019– 2019

[16] Steichen M, Fiz B, Norvill R, Shbair W, State R (2019) Blockchain-Based, Decentralized Access Control for IPFS. IEEE Access pp 15–15

[17] Tar A (2018) Proof of Work, Explained. Cointelegraph. Accessed on: Apr 3, 2021. [Online]. https://cointelegraph.com/explained/proof-of-work-explained

[18] Wang S, Zhang Y, Zhang Y (2018) A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. IEEE Access pp 29–29

[19] Zheng W, Zheng Z, Chen X, Dai K, Li P, Chen R (2019) NutBaaS: A

Blockchain-as-a-Service Platform. IEEE Access 7(10):134422–134433, DOI 10.1109/access.2019.2941905, URL https://dx.doi.org/10.1109/access.2019. 294190