MIDDLEWARE FRAMEWORK FOR DISTRIBUTED CLOUD STORAGE

By

Matthew B. Hancock

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
Carlos Varela, Thesis Adviser

_____
Christopher Carothers, Member

_____
Ana Milanova, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2015
(For Graduation May 2015)

CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGMENT vi

ABSTRACT vii

1. INTRODUCTION 1

    1.1 Motivation 1

    1.2 Structure of Thesis 2

2. CLOUD STORAGE PROVIDER LIMITATIONS 3

    2.1 Increased Cost 3

    2.2 Limited Bandwidth 4

    2.3 Decreased Security 4

    2.4 Unpredictable Availability 4

3. CONCURRENT CLOUD STORAGE MODEL 6

    3.1 Python with Cloud Storage 6

        3.1.1 Diverse User Base 6

        3.1.2 Ease for Developers 6

    3.2 Expandable Framework 7

4. DISC 10

    4.1 Approach 10

    4.2 User Policy Model 11

        4.2.1 Millionaire Policy 11

        4.2.2 Security Policy 11

        4.2.3 Availability Policy 12

        4.2.4 Budget Policy 14

        4.2.5 Low Replication Policy 14

    4.3 Data Throughput Management 15

        4.3.1 Approach 15

        4.3.2 Throughput Model 16

4.3.3 Migration Procedure 17
4.3.3.1 Migration Rules 17
5. APPLICATION AND EXPERIMENTAL RESULTS 19
5.1 Experiment Setup 19
5.2 Initial Experimental Results 20
5.3 Experimental Results 21
5.3.1 Case 1 - Security Minded 21
5.3.2 Case 2 - Availability Minded 23
5.3.3 Case 3 - Budget Minded 25
5.3.4 Case 4 - Scaling File Size 27
5.3.5 Case 5 - Bandwidth Bottleneck 28
5.3.6 Case 6 - Using a Manager 29
6. RELATED WORK 33
6.1 RACS 33
6.2 HAIL 33
6.3 Hybrid Cloud Storage 34
7. DISCUSSION 35
7.1 Conclusion 35
7.2 Future Work 36
LITERATURE CITED 37

LIST OF TABLES

2.1 Storage Providers Statistics as of 2014 3

3.1 Lines of Code Comparison 7

LIST OF FIGURES

3.1 Pseudo Code for Driver Class 8

3.2 Pseudo Code for Abstracted Base Storage Class 8

4.1 DISC File Distribution 10

4.2 Splitting a File Using Security Policy 12

4.3 Availability Policy Experiment 13

4.4 Comparison of Replication Processes 15

4.5 Manager Model 16

4.6 Migration Procedure Model 17

5.1 Uploading Using a Single Provider 20

5.2 Downloading Using a Single Provider 20

5.3 Security Minded Policy Uploading 22

5.4 Security Minded Policy Downloading 22

5.5 Availability Minded Policy Uploading 24

5.6 Availability Minded Policy Downloading 24

5.7 Budget Minded Policy Uploading 26

5.8 Budget Minded Policy Downloading 26

5.9 Upload Time Saved with DISC 27

5.10 Varying Number of Cloud Providers 29

5.11 Data Throughput Uploading 30

5.12 Time Difference Between Extreme Ranges 31

ACKNOWLEDGMENT

I would like to thank my advisor, Professor Carlos A. Vaerla, for his time and effort in supervising my research. His availability and willingness to discuss has made this a great learning experience. Many thanks to the colleagues I have worked with at the Worldwide Computing Lab and to those that have given me support and advice throughout this journey. Lastly and most importantly, a great thanks to the support of my family for their everlasting encouragement.

ABSTRACT

The device people use to capture multimedia has changed over the years with the rise of smartphones. Smartphones are readily available, easy to use and capture multimedia with high quality. While consumers capture all of this media, the storage requirements are not changing significantly. Therefore, people look towards cloud storage solutions. The typical consumer stores files within a single provider. They want a solution that is quick to access, reliable, and secure. Using multiple providers can reduce cost and improve overall performance. We present a middleware framework called *Distributed Indexed Storage in the Cloud (DISC)* to improve all aspects a user expects in a cloud provider. The consumer provides the middleware files, which get processed through user policies, and stored within the cloud. The process of uploading and downloading is essentially transparent. The upload and download performance happens simultaneously by distributing a subset of the file across multiple cloud providers that it deems fit based on policies. Reliability is another important feature of DISC. To improve reliability, we propose a solution that replicates the same subset of the file across different providers. This is beneficial when one provider is unresponsive, the data can be pulled from another provider with the same subset. Security has great importance when dealing with consumer's data. We inherently gain security when improving reliability. Since the file is distributed using subsets, not one provider has the full file. In our experiment, performance improvements show when delivering and retrieving files compared to the standard approach. The results are promising, saving upwards of eight seconds in processing time. With the expansion of more cloud providers, the results are expected to improve.

This abstract is to appear in: M. B. Hancock and C. A. Varela, "Augmenting Performance For Distributed Cloud Storage," *in Proc. of the IEEE/ACM 15th Int. Symp. on Cluster, Cloud and Grid Computing*, 2015.

1. INTRODUCTION

1.1 Motivation

The mobile phone market has continued to grow rapidly with no indication of slowing down. Followed by browsing the web, the most popular features on a smartphone are the camera and video capabilities. The storage space on a smartphone to hold these multimedia files is very limiting compared to a computer or cloud storage. This limitation influences users to look toward cloud storage options. Social psychologists looked at why cloud storage is growing in popularity [1]. They note the simplicity and ease of use, the ability to share and communicate with friends, and the availability of their files from anywhere with an internet connection. These features empower a user to be more productive.

Cloud storage is great on paper, but has its own limitations as well. Files stored with a cloud provider are susceptible to compromised security, have poor network performance during the upload or download process, and are only accessible when the provider is operating. The problem that arises is how can we produce a more secure, more efficient, and fault tolerant approach while using multiple storage providers and leveraging a concurrent model. A distributed cloud storage approach provides an elegant solution that is transparent to the user.

There are a number of techniques that tackle only a portion of the cloud provider's shortcomings [2]. The majority of these techniques are geared toward the enterprise market, not allowing the consumer market to take advantage of the technologies. We introduce a middleware framework we call *Distributed Index Storage in the Cloud (DISC)*. The main backbone behind DISC is a set of user defined policies that enable DISC to operate differently and accommodate various user preferences. The various user policies respectively tackle the shortcomings depending on a certain budget, amount of security, and availability a user requires.

1.2 Structure of Thesis

The structure of this thesis is as follows: In Chapter 2, we begin with an overview of Cloud Storage Providers and their drawbacks. Following in Chapter 3, we introduce the concurrent model for cloud storage and how it is used within this framework. In Chapter 4, we take a look at how to combine cloud storage and how to intelligently process the data through user policies. Next in Chapter 5, we examine our approach through different case scenarios with various storage providers. In Chapter 6, we review some related work being done in this field. Lastly, Chapter 7 will recap and discuss some future work to conclude this thesis.

2. CLOUD STORAGE PROVIDER LIMITATIONS

Users by nature see cloud storage as a means for reliable backups, sharing of data, twenty-four hour access, and synchronization across multiple devices. As seen in Table 2.1, a typical free account offers a small amount of storage space. Many users have accounts with multiple providers to avoid paying for more storage space. Utilizing a single cloud provider's resources proves to have many drawbacks: increased cost, limited bandwidth, decreased security, and unpredictable availability.

Table 2.1: Storage Providers Statistics as of 2014

Source	Free Storage	Paid Storage
Dropbox [3]	2GB	1TB $9.99/month
Box [4]	10GB	100GB $10.00/month
Google Drive [5]	15GB	100GB $1.99/month 1TB $9.99/month
One Drive [6]	15GB	100GB $1.99/month 200GB $3.99/month 1TB $6.99/month

2.1 Increased Cost

The key reason to use cloud storage is the amount of extra space it provides. However, the provided free accounts are well below the sufficient amount of space needed to store files. On average, one photo requires 5MB of disk space [7] and a three-minute high-definition video requires 500MB [8]. Multimedia files are among the most popular to be stored in the cloud because they provide hassle free backup and easy sharing capabilities. Depending on the type and quantity of data, paid accounts are necessary for cloud storage to be of any service, by increasing storage

This chapter is to appear in: M. B. Hancock and C. A. Varela, "Augmenting Performance For Distributed Cloud Storage," *in Proc. of the IEEE/ACM 15th Int. Symp. on Cluster, Cloud and Grid Computing*, 2015.

space. Budget must now be considered when using cloud storage. In Table 2.1, we see that the paid accounts can be costly year-over-year. The demand to increase communication bandwidth and maintenance to cushion the impact of occasional outages, add yearly costs. As demands increase, it is likely cost will follow [9].

2.2 Limited Bandwidth

A major source of performance problems for cloud services is the communication time between the client computer and the web server in the cloud [9]. The bandwidth to upload or download files from the cloud are throttled to help improve overall performance among all active users. For instances, a user has 1.5GB of data stored with Dropbox who throttles the network bandwidth to 1.5MB/s. This would require 17.06 minutes assuming full bandwidth throughout the entire download process. Chapter 4 describes how a user's local network can be fully utilized through DISC and its concurrent cloud approach.

2.3 Decreased Security

Storage providers are getting hacked more frequently than ever before. With many enhancements in the security and firewall protection, simple attacks like phishing or brute force can lead to unauthorized access to millions of user's sensitive information. Many attacks are blocked, but a few are granted unauthorized access to user accounts and files. Storing all files in one location, in an unencrypted form for some providers, shows to be a security flaw. In Chapter 4, we propose an approach to tackle this exact problem for those security-minded users.

2.4 Unpredictable Availability

There are two kinds of outages known as permanent and temporary. A permanent outage is when a cloud provider goes out of business while a temporary outage occurs when the cloud provider is unavailable for a span of time [9]. This thesis refers to temporary outages.

Twenty-four hour access is the ideal expectation, but unanticipated errors and outages are likely to occur. The availability of cloud storage includes persistent

runtime and recovery. High availability is needed to ensure application quality of service (QoS) [10]. A simple outage can occur from a network problem or a database corruption problem. Both cases prevent a user from accessing or creating new files. During one incident in August 2014 [11], Microsoft encountered a problem with their Visual Studio Online Service that resulted in five-and-a-half hours of outage time. This instance hinders users' productivity worldwide. Throughout Chapter 4, we see how fault tolerance is used to recover during a service outage or a file corruption.

3. CONCURRENT CLOUD STORAGE MODEL

Combining storage providers, as if they are one unit, can provide an elegant solution and mitigate many of the limitations. Concurrent cloud storage is defined as accessing and processing data through multiple cloud storage providers in parallel. Throughout the various test cases in Chapter 5, we will see that this concurrent model performs better when more storage providers are linked.

3.1 Python with Cloud Storage

The benefits for using Python in developing the concurrent cloud storage is the large user base that Python can run on and the ease for developers to configure this framework to their needs.

3.1.1 Diverse User Base

Cloud storage is used throughout all industries from the consumer to the enterprise markets. The framework had to be built on a language that is capable of performing well for network performance as well as run on most machines. Whether it is a Windows, Mac, or Linux, Python can be run on these platforms. In addition to compatibility with the machine software, the framework had to interface well with currently existing public clouds. Table 2.1 lists a few public cloud providers that interface well with Python. Considering these requirements, Python proved to be a reasonable choice.

3.1.2 Ease for Developers

There are multiple languages that allow concurrent processes such as Python, Ruby, and Java. Among the ones listed, Java is the most popular, but more cumbersome to develop with. Python is the next to follow, increasing year-over-year in popularity. With its less complex development, allowing shorter lines of code, Python is a suitable solution for creating a concurrent model.

Table 3.1: Lines of Code Comparison

Source	Language	Lines of Code
Dropbox	Python	30
	Ruby	30
	Java	54
Google Drive	Python	37
	Ruby	41
	Java	57

Table 3.1 compares the lines of code for a simple upload and download application written for two storage providers in three languages. Python shows to be the shortest lines of code to develop, making it easier to maintain for future updates.

3.2 Expandable Framework

There are many choices that exist to store files, whether in a private cloud or a public cloud. The framework needs to accommodate both types. The best approach to achieve this is through abstracted classes. The base class or the parent class, holds common information that all storage types will need such as any required functions. Each separate storage provider will need its own class, holding specific information. Using abstracted classes allows the driver class to call the same functions, where each function performs a different procedure. The relationship between classes is shown in Figures 3.1 and 3.2 with some pseudo code.

```
def main(arguments):
    # initializing all storages and execute login function
    services = []
    services.append(DropboxStorage())
    services.append(GoogleStorage())
    services.append(...)

    # create user profile and run the input file
    # through the policies to get newly created files
    user = UserPolicies()
    fileNames = user.<policy>

    # start all services
    for service in services:
        service.setupProcedure(...)
        service.start()

    # wait for all threads to finish
    for service in services:
        service.join()
```

Figure 3.1: Pseudo Code for Driver Class

```
class BaseStorage(threading.Thread):
    def __init__(self):
      # initialize the thread
      super(BaseStorage, self).__init__()

    def setupProcedure(self, upload, args):
      # prepare execute command

    def executeProcedure(self):
      # call specified command

    @abstractmethod
    def login(self):
      pass

    @abstractmethod
    def setupLogin(self):
      pass

    @abstractmethod
    def upload(self, fileLocation):
      pass

    @abstractmethod
    def download(self, fileLocation):
      pass
```

Figure 3.2: Pseudo Code for Abstracted Base Storage Class

In Figure 3.1, note that both classes `DropboxStorage()` and `GoogleStorage()` are child classes from their parent `BaseStorage`. Each storage provider is initialized through the `setupProcedure` method where the execute statement is setup using a command and its arguments. When the thread is started, the `executeProcedure` function is called. It is quite easy to add new storage providers by simply creating a new child class and adding the new class to the array `services[]`.

Looking at Figure 3.2, any child class must override any function tagged as `@abstractmethod`. All children inherently call the setup and execute functions due to the abstracted classes. Due to inheritance, there is less code for developers to write and handle, which makes it simpler to expand the list of cloud providers.

4. DISC

4.1 Approach

We developed Distributed Indexed Storage in the Cloud (DISC) to be an expandable framework that follows the concurrent model for communicating and interfacing with cloud storage providers.

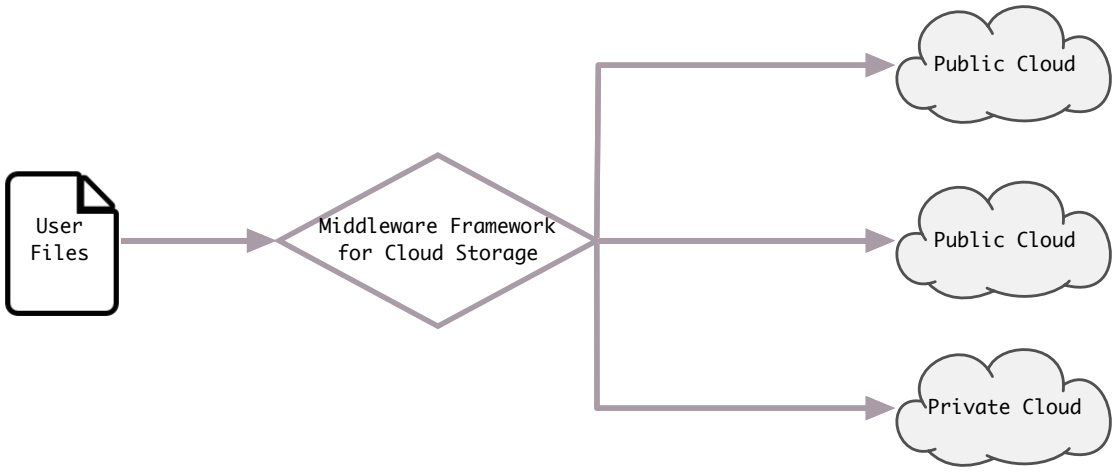

Figure 4.1: DISC File Distribution

With each given file, DISC will follow a set of user policies to mitigate some limitations to using a single cloud storage provider. Security concerns can be resolved through splitting a file on a byte level. Space constraints can be tackled through compression and low replication. Increased availability can be achieved by separating images on a pixel level. These policies can be used to configure how DISC works for different scenarios and preferences.

This chapter is to appear in: M. B. Hancock and C. A. Varela, "Augmenting Performance For Distributed Cloud Storage," *in Proc. of the IEEE/ACM 15th Int. Symp. on Cluster, Cloud and Grid Computing*, 2015.

4.2 User Policy Model

This section looks at several user policies that can be used with DISC to create a more dynamic model. These policies include: millionaire, security, availability, budget, low replication, or a combination.

4.2.1 Millionaire Policy

Users want to access their files one-hundred percent of the time. However, this does not come without a cost involved. This model is based on a user who is willing to spend extra money in return for more space. With the extra space, we replicate the files across all cloud storage providers. Higher replication increases the probability the requested file will be accessible from one of the n providers. In Figure 4.4, we visually compare the differences between full replication and partial replication.

4.2.2 Security Policy

Moving data from the local source to a public cloud opens the possibility for data to be vulnerable and intercepted during transfer. DISC takes an approach where every i^{th} byte is stored with a different cloud provider. Splitting a file is determined using the following equation:

$$CSP_i = F_{byte}, \qquad i = F_{byte} \bmod CSP_{total} \tag{4.1}$$

This controls which byte of the original file F_{byte} is stored with which cloud storage provider CSP_i. Each byte location is calculated based on the total number of cloud storage providers CSP_{total}. Figure 4.2 gives a visual representation of how DISC splits a file based on equation 4.1.

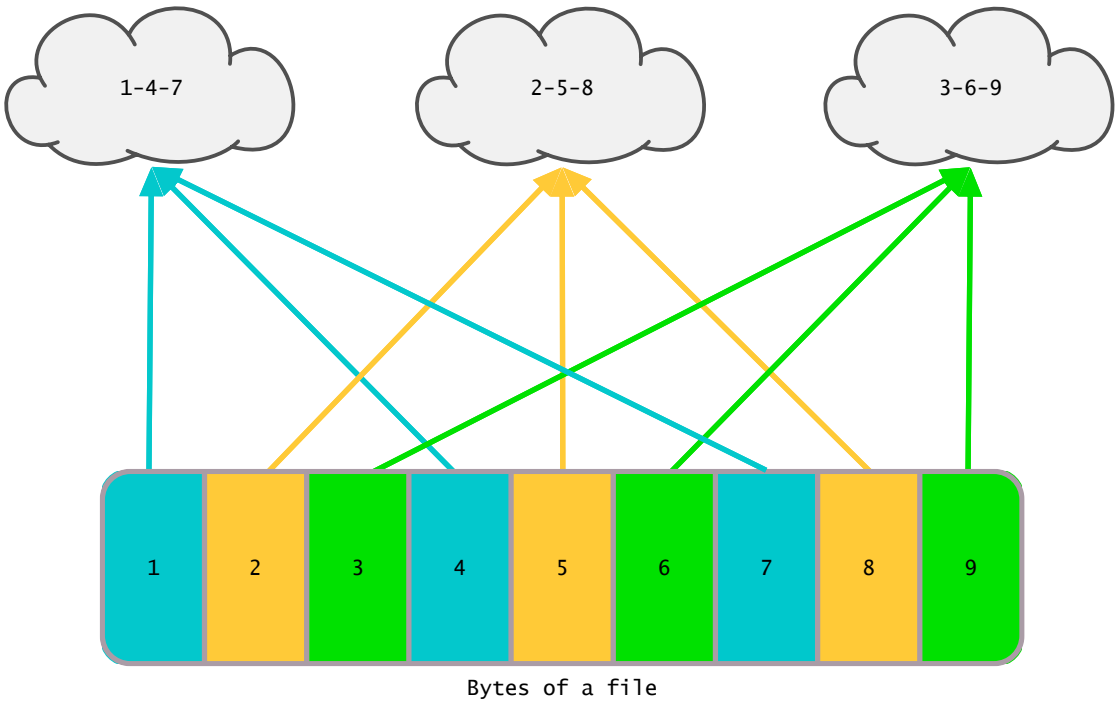

Figure 4.2: Splitting a File Using Security Policy

When unauthorized access occurs, the hacker is typically able to open and view the files. However, DISC creates equal size files that are unreadable as separate entities. Only when combined in the correct order is the original file again readable.

4.2.3 Availability Policy

Performance is an important factor to the user experience. To improve performance, a file must be split into multiple files now rendering each split file unreadable. The availability policy separates the files, similar to 4.1, while creating usable and viewable partitions. This policy currently only works with image files. Figure 4.3 shows an experiment using two cloud providers implementing the availability policy.

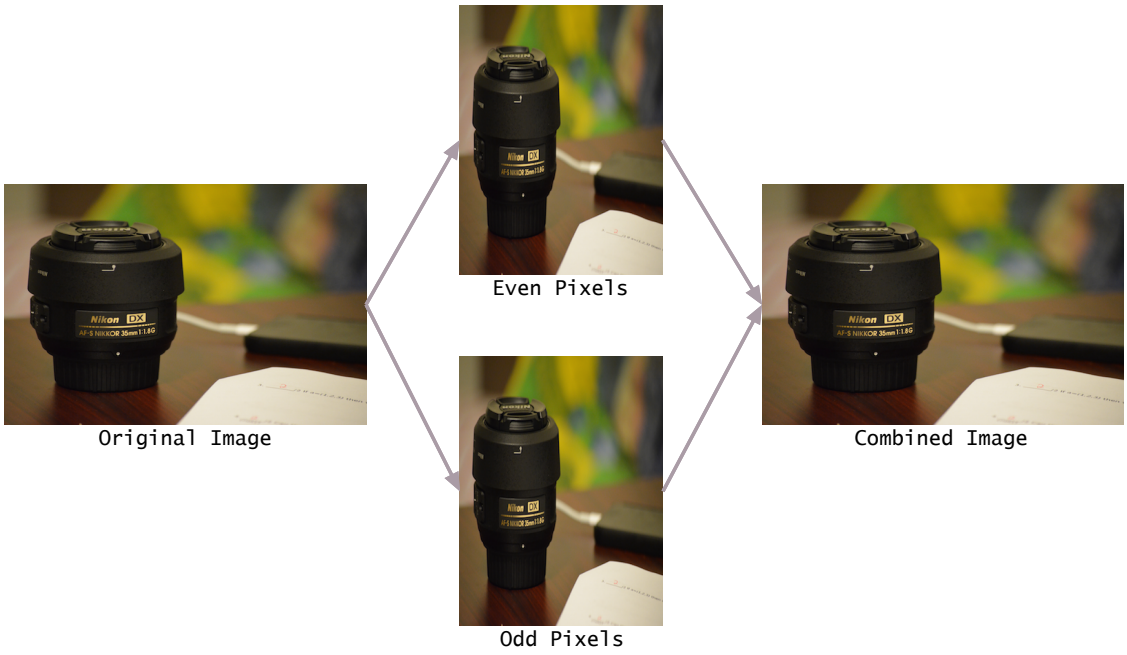

Figure 4.3: Availability Policy Experiment

Taking the even and odd pixels from the original image creates two new images. The user will always see the combined image when both providers are operating. In case one provider fails to operate, the image is still viewable in its original aspect-ratio. To achieve the same aspect-ratio, we replace unreachable portions of the image with pixels in close proximity. To improve performance, we do not calculate an average between two pixels, but rather simply replace the lost data with an exact replica from a different portion. As seen in the experiment above, if the even pixels were lost, the odd pixels would be duplicated to fill in lost pixels outputting a correct aspect-ratio.

Increasing the total number of cloud providers will produce better results. In most cases, only one provider will be nonfunctional. Simple math shows that as the denominator approaches infinity, the viewable image approaches one-hundred percent. The percentage P of viewable image can be determined with the following equation where CSP_{online} is the number of operating providers:

$$P = \frac{CSP_{online}}{CSP_{total}} \times 100 \tag{4.2}$$

For example, five cloud providers with one not operating, results in eighty percent of the original image with the lost pixels duplicated by an adjacent set of pixels.

4.2.4 Budget Policy

More storage space is available, but only at a premium cost. This policy tackles the problems for a user with limited space. DISC will compress the file until reaching a specified tolerance. Document files cannot be compressed significantly, so we look to compress multimedia files. With each compression, DISC will alter the file size by half. This results in the image width and height dimensions to be a quarter smaller. Unlike other policies, this results in a permanent file reduction. There is no reason to uncompress and return the original image.

4.2.5 Low Replication Policy

Rather than replicating the files among all storage providers, this policy calculates the number of replicas to create using a subset of the total storage providers. Replication is a valuable feature, but their budget may be constrained. We can now increase the probability the file is accessible, while reducing the total storage space. The number of replicas R is determined using the following equation:

$$R = \left\lfloor \frac{CSP_{total} - 1}{l} \right\rfloor \tag{4.3}$$

The file must be stored in at least one location. Therefore, we subtract one from the total cloud providers since we cannot make a replica on the same storage provider. In the equation, l is the level describing how important replication is to the user. A lower number will produce more replicas while a higher number will reduce the number of replicas. Figure 4.4 gives a comparison between the millionaire policy and the low replication policy.

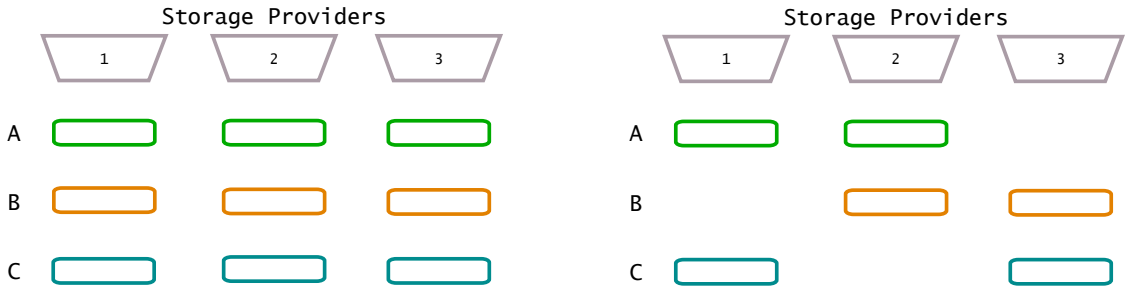

Figure 4.4: Comparison of Replication Processes

Figure 4.4 represents the final result from replication using the millionaire policy (left) versus the low replication policy with $l = 2$ (right). There are three files labeled A, B, C that need to be stored among three storage providers.

Replication among multiple storage providers has benefits that users find important. A file can be accessed from a separate provider if one provider is experiencing an outage. When a corrupted file is detected, DISC can recover the file from another provider. The more replications, the higher probability for accessibility and recoverability. Although there is a cost for more storage, the user will gain more replication through the millionaire policy. In contrast, the low replication policy gives the benefits from replication with no additional cost.

4.3 Data Throughput Management

4.3.1 Approach

During an upload or download, bandwidth is constantly changing, which can enhance or hinder the time taken to process the request. Using a feedback loop, we can alter which cloud providers will receive more of the data and which will receive less. After each iteration, calculations are performed to determine the throughput. If the difference in throughput between cloud providers falls below a threshold, DISC will alter the 1:1 ratio to be skewed. The cloud provider that performs better, will receive more data to process while the contrasting provider receives less data. This alteration will level out extreme cases where one cloud provider is performing poorly, requiring excess throughput times. We use the models depicted in Figure 4.5 and Figure 4.6 to manage and improve the changing bandwidth performance.

4.3.2 Throughput Model

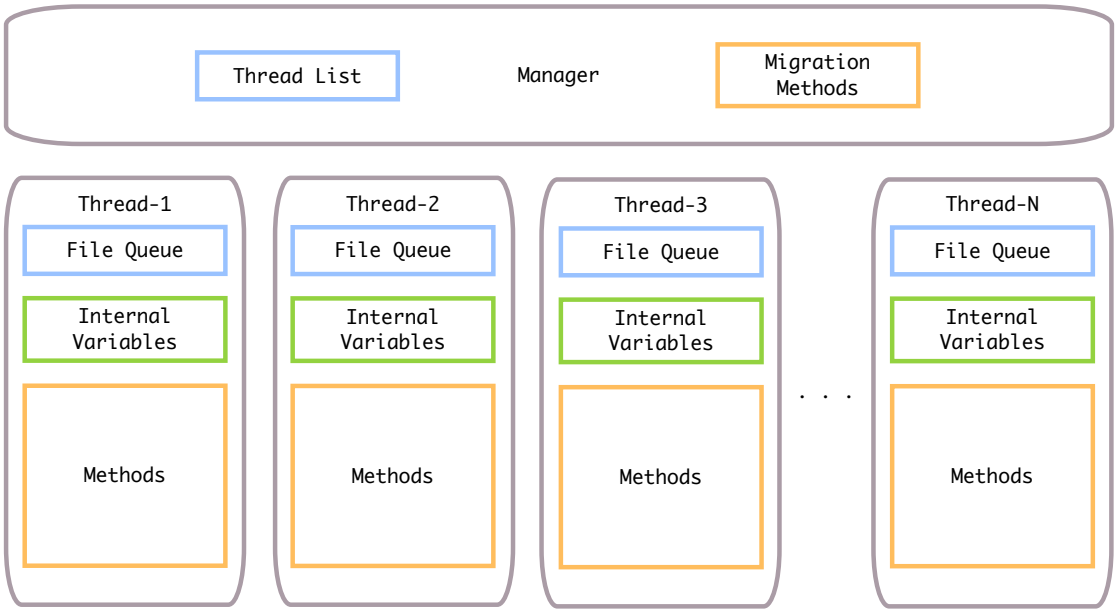

Figure 4.5: Manager Model

Figure 4.5 depicts the structure of a Manager and its relationship with an unknown number of cloud providers, shown as threads. Each cloud provider gets its own thread container, encapsulating multiple pieces of information shown as a file queue, internal variables, and methods. A file queue simply holds a list of filenames to be uploaded or downloaded. The file queue is accessible to the Manager to allow migration of files between threads. Internal variables keep track of information such as number of files processed, time taken to process the file, and a Manager object. All this information is accessible to the Manager, but cannot be changed outside of the thread container. Lastly, there are multiple functions for populating the file queue, calculating ratios, and private functions aiding in processing a file.

Supervising all these threads is a Manager. Within the Manager is a list of threads and methods to facilitate the migration procedures. When downloading, the Manager takes a proactive approach by predicting accurate forecasts of dynamically changing performance characteristics from a distributed set of resources [13]. Using prior statistics, we will upload data placing more files in storage providers that we expect will download faster. In contrast, the Manager takes a reactive approach

during the upload process. We can use real-time information rather than predicting changing performance. The Manager is called upon, each time a file is processed. At this time, statistics from each thread are gathered followed by calculations to determine whether one cloud provider should gain or lose a file. To improve performance, there are three times as many files as cloud providers. The extra files, which are of equal size, allow the Manager to detect early on if one cloud provider has slow performance which will cause a migration procedure to occur.

4.3.3 Migration Procedure

A migration procedure is the process of altering the responsibility of a cloud provider by removing one file from the file queue and transferring this file to another file queue for a different thread. A migration only occurs if the migration rules are satisfied.

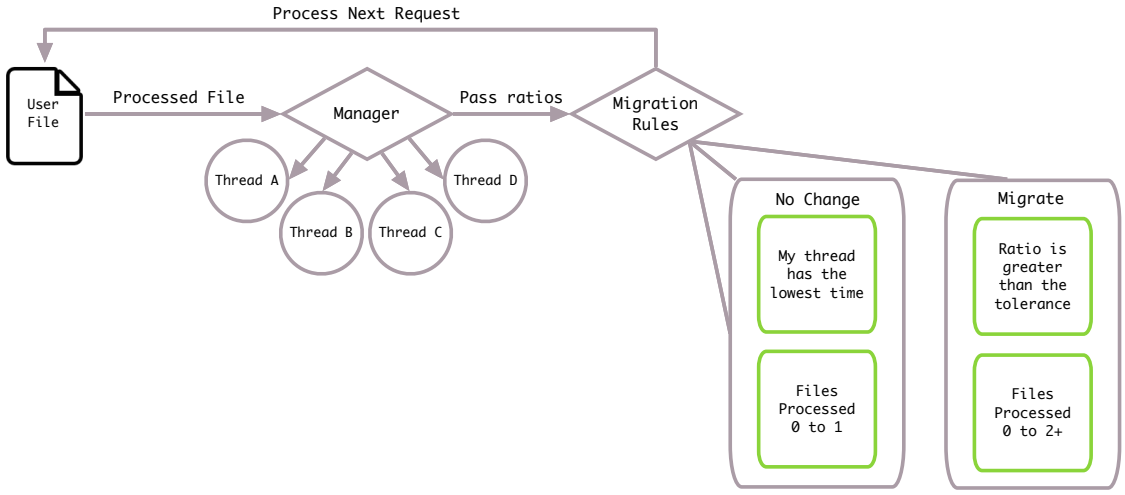

Figure 4.6: Migration Procedure Model

4.3.3.1 Migration Rules

As seen in Figure 4.6, there are four migration rules that help determine if a migration should occur or keep all threads unchanged. These decisions are based on two statistics stored within each thread. These statistics include the number of files processed and the time taken to process the most recent file. We use the most recent file as it best represents the thread's performance.

There are two scenarios that cause the Manager to leave all thread containers unchanged. The first occurs when the thread with the lowest processing time and the thread that called the Manager, are the same. No improvements can be made knowing that thread is performing the fastest. This will more likely occur when threads start to exit as the duration of the program increases. The next scenario uses statistics from the number of files processed. Completing a file increases the number of files processed by one. However, if the lowest number of files processed is zero, there is not enough information to give a good indication that the other provider is performing slowly. From this information, we can only infer that the thread finished first and the other threads may or may not finish close behind.

File migration occurs during two similar scenarios. The first scenario leverages the time taken to process the most recent file from each thread. The minimum time among all threads is compared to the time taken from the thread who called the Manager. The difference between these two times must be greater than a tolerance level. This tolerance level is set at 1.5 seconds. It is not advantageous to migrate if the difference is too small. The next scenario occurs when the lowest number of files processed is zero and the thread that called the Manager has completed processing two or more files. In contrast to before, we can confidently say we need to migrate. Having a 0:2+ ratio means one provider has completed two files in the same time another provider has completed zero. This is a strong indication that one provider is operating at a subpar performance.

These rules are used to determine when and where to migrate a file. With constant changes in bandwidth performance, we can now improve the time taken to process files in either the upload or download process.

5. APPLICATION AND EXPERIMENTAL RESULTS

5.1 Experiment Setup

In the multiple experiments for uploading and downloading files, we use two public clouds Dropbox and Google Drive. The computer is connected via WiFi 802.11n. The network performance is tested as 68.86Mbps download and 12.45Mbps upload. It is important to note, with high local network speeds, some public clouds throttle this speed.

Throughout the various tests, we use a 20.9MB image file. The purpose for a high resolution image is to compare overall quality with performance of splitting or rescaling the image. Rescaling an image is tackled using the Python library, scikit-image [12]. We have two instances of Dropbox and two instances of Google Drive during the experiment.

A portion of this chapter is to appear in: M. B. Hancock and C. A. Varela, "Augmenting Performance For Distributed Cloud Storage," *in Proc. of the IEEE/ACM 15th Int. Symp. on Cluster, Cloud and Grid Computing*, 2015.

5.2 Initial Experimental Results

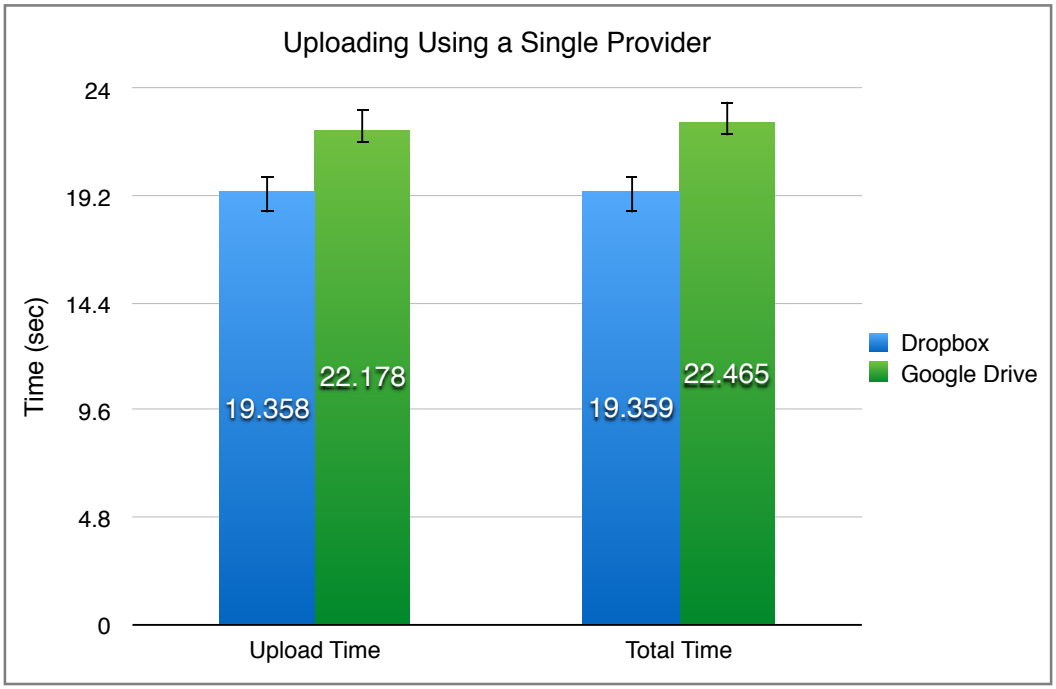

Figure 5.1: Uploading Using a Single Provider

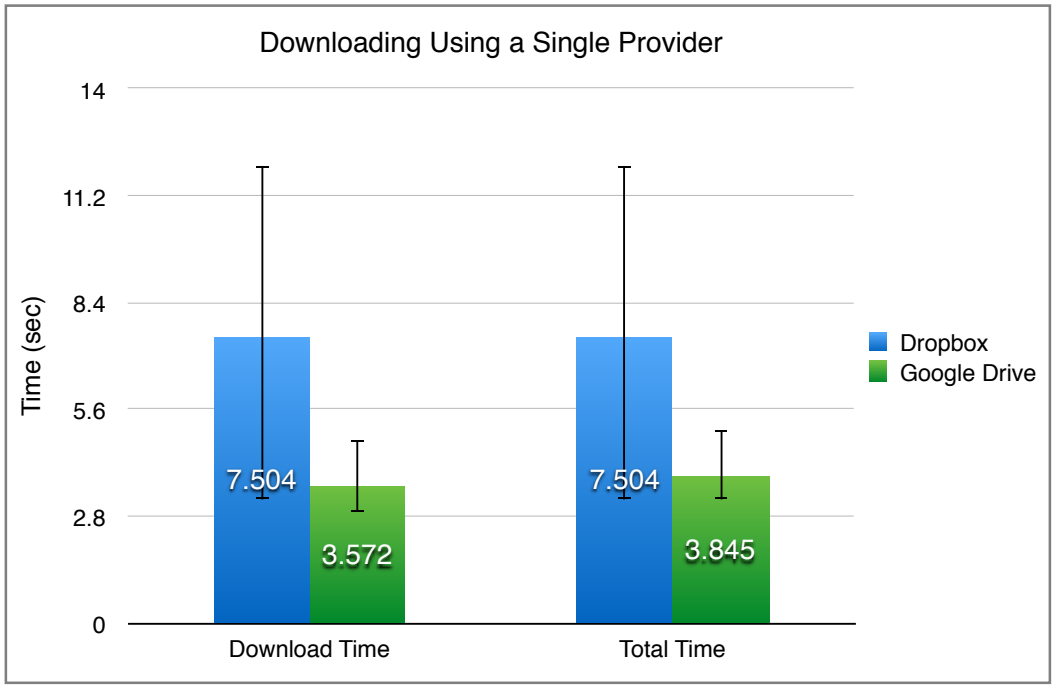

Figure 5.2: Downloading Using a Single Provider

From Figures 5.1 and 5.2 respectively, we now have starting data for the amount of time it takes to interface with the public cloud providers. The first set of column bars is the time to upload or download the image file. The second set of column bars is the complete time which includes the login process, uploading, and stopping the thread. It becomes clear that the time taken to login and create or destroy a thread is negligible.

We can see that both Dropbox and Google Drive do not behave similarly in either experiment. Both cloud providers operate with similar performance during the upload process. However, we can see a tremendous difference in the download process concluding that cloud providers put more emphasis on download performance. We also see Dropbox has a large range of download times. Factors for these differences are not covered in this thesis, but is important information to consider.

5.3 Experimental Results

5.3.1 Case 1 - Security Minded

In this test experiment, we uploaded and downloaded the image file using the security policy. Using Figure 5.3 and Figure 5.4, we can see the benefits and minor drawbacks that result from this policy.

Notice that in both cases, the sequence in which each storage provider completes processing, occurs in no specific order. For this reason, the total upload or download time is seen as the maximum time from all providers. We can see that all instances require similar amounts of time to complete for downloading in Figure 5.4. However, one instance during the upload process took noticeably longer compared to other instances. This could be caused by the bottleneck in bandwidth or unknown circumstances with the cloud provider.

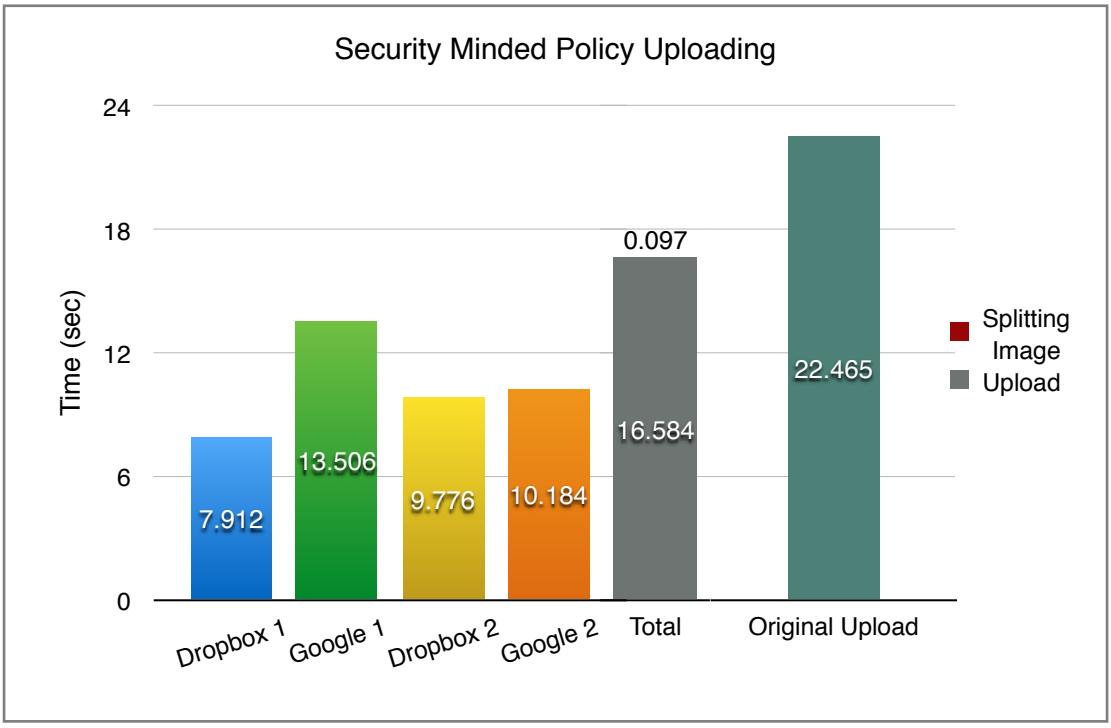

Figure 5.3: Security Minded Policy Uploading

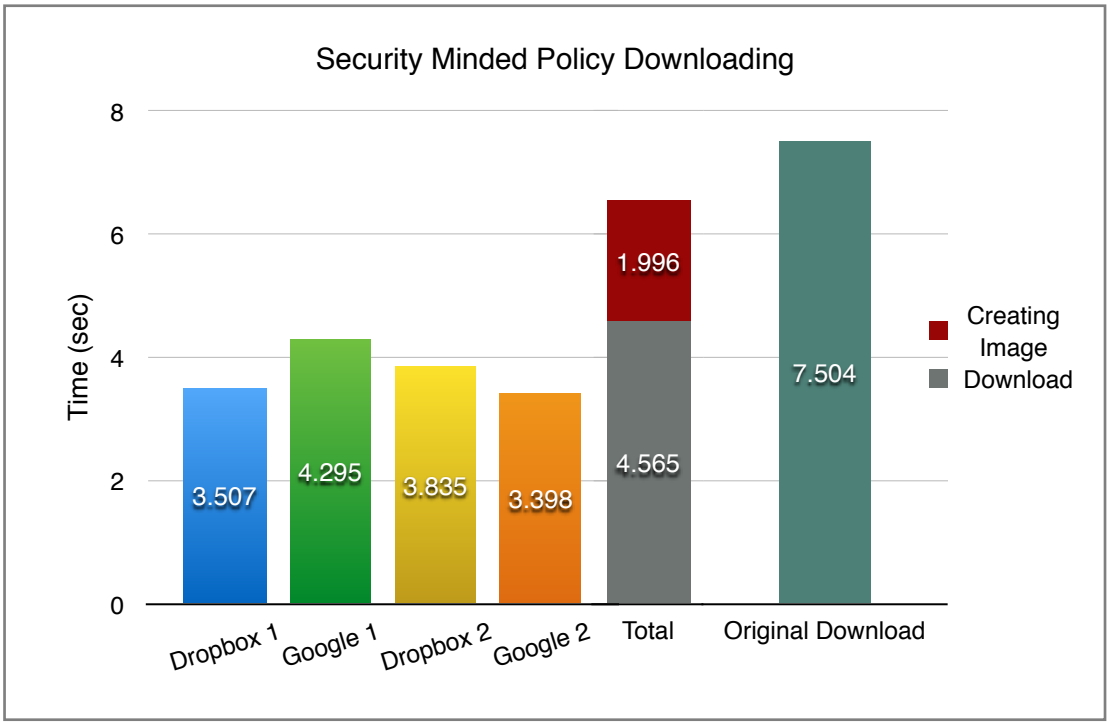

Figure 5.4: Security Minded Policy Downloading

By altering the number of cloud providers, we can see how DISC benefits by processing requests concurrently. Comparing Figure 5.3 with our initial experiment, there is a 34.67% decrease in upload time. With this improved performance, we were able to increase the level of security for each file. Comparing Figure 5.4 with our initial experiment as well, there is a 12.56% decrease in download time. The majority of this time is used to download all files. The process of correctly ordering the files to reproduce the original file is very quick. Removing the amount of time used for concatenating, we can see there is a 64.38% decrease in download time. These results prove two aspects: We gain significant improvements in security and upload performance. We also see a small improvement in download performance with a significant increase in security.

5.3.2 Case 2 - Availability Minded

In this experiment, we uploaded and downloaded the image file using the availability policy. Using Figure 5.5 and Figure 5.6, we can see the benefits and drawbacks that result from this policy.

In both cases, there is a significant amount of time used to process the image. Looking farther into the processing time, it is mostly occupied with saving the image file. We can note again, that each storage provider completes its tasks, in no specific order. This behavior is seen throughout all experimental tests. This test appears to follow similar patterns as our initial experiment in Figure 5.1. Both Google instances required twice as much time as Dropbox to complete their uploads.

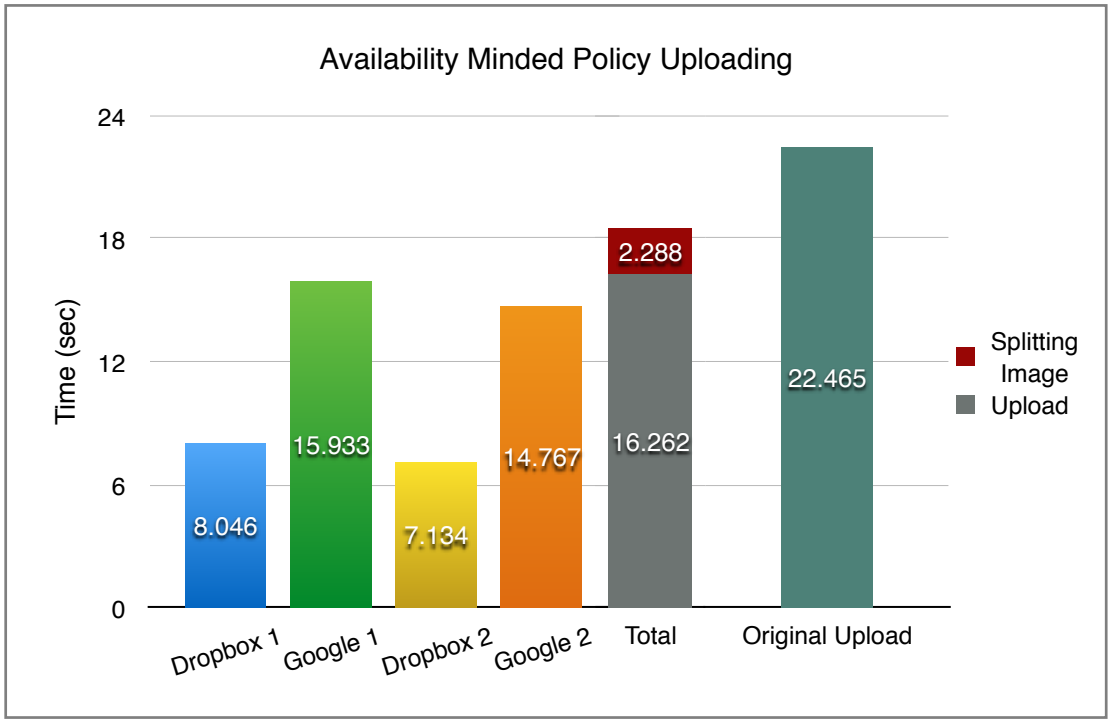

Figure 5.5: Availability Minded Policy Uploading

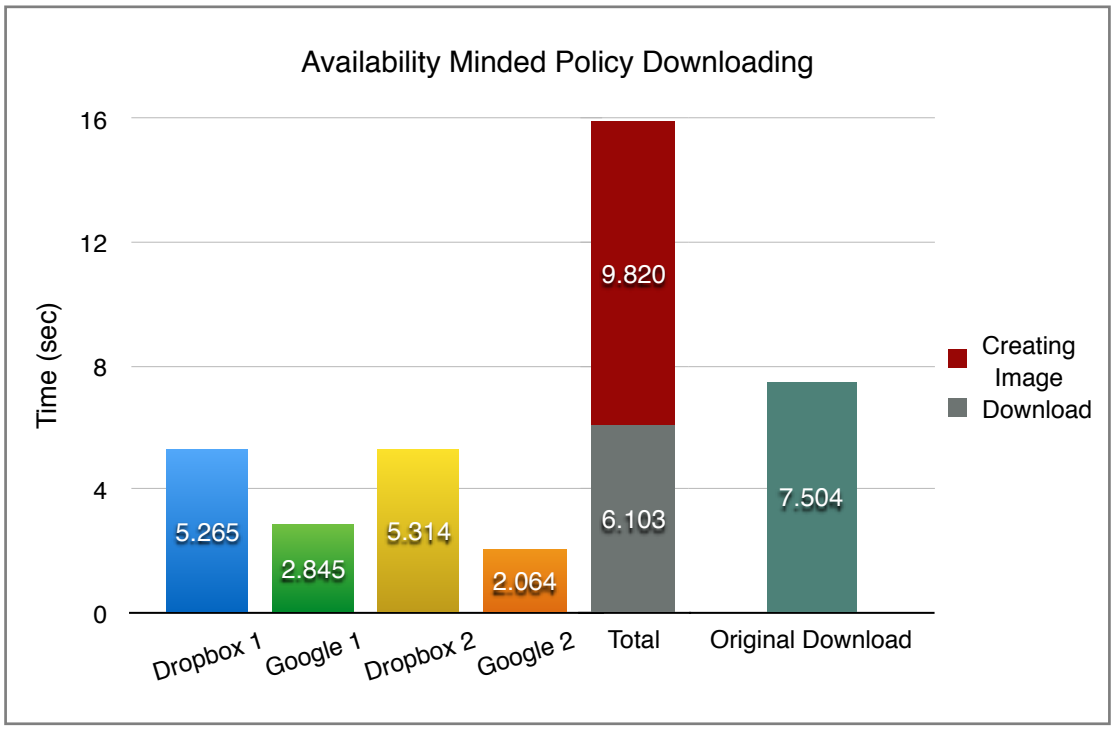

Figure 5.6: Availability Minded Policy Downloading

Removing the time required to process the image, we can see that DISC has improved performance in the upload and download times. DISC is able to upload all four images 38.14% faster and download the same four images 22.95% faster when compared to the initial results in Figure 5.1 and Figure 5.2 respectively. Looking at the total time required, upload time decreases by 16.36%. The Security Minded experiment required much less time to split the image because it is working at the byte level, producing four unviewable images. In this experiment, we are working at the RGB level which preserves the three color values for each pixel. While this process requires more time, roughly two seconds, is nonetheless fast. We see the same observation with downloading the four images as in the upload experiment. Of the total time, 61.67% of the that time is used for image processing resulting in an increased download time of 112.20%. While this policy takes a small hit on performance, it gains great improvements for the availability of the file.

5.3.3 Case 3 - Budget Minded

In this experiment, we uploaded and downloaded an image file using the budget and security policies. The budget policy is used to rescale the image file and the security policy splits the image into four equal parts so we can concurrently upload and download the files. The budget policy removes a quarter pixels in each dimension, width and height, resulting in a fifty percent file reduction. Therefore, the new image is compressed to 10.9MB in size. Using Figure 5.7 and Figure 5.8, we can see the benefits that result from combining these policies.

This policy requires image processing similar to the availability policy. In contrast to the availability policy, rescaling the image requires significantly less time. Looking at the individual download times, they appear to be more similar than previous experiments. As download times decrease, all instances being to complete in roughly the same time.

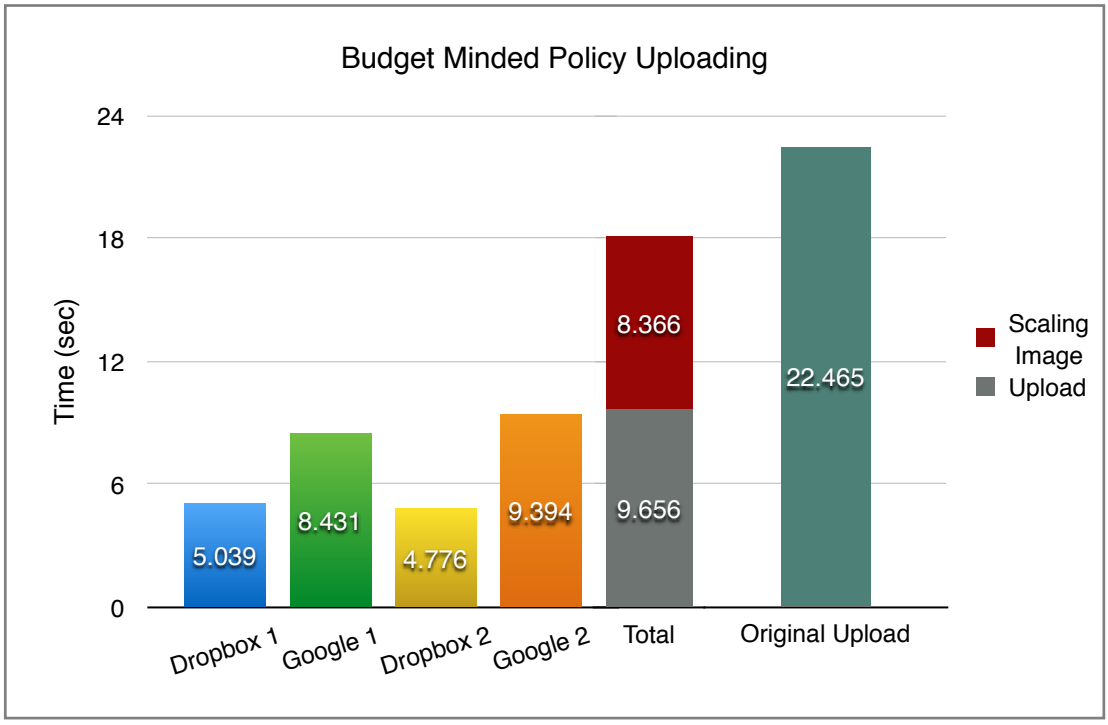

Figure 5.7: Budget Minded Policy Uploading

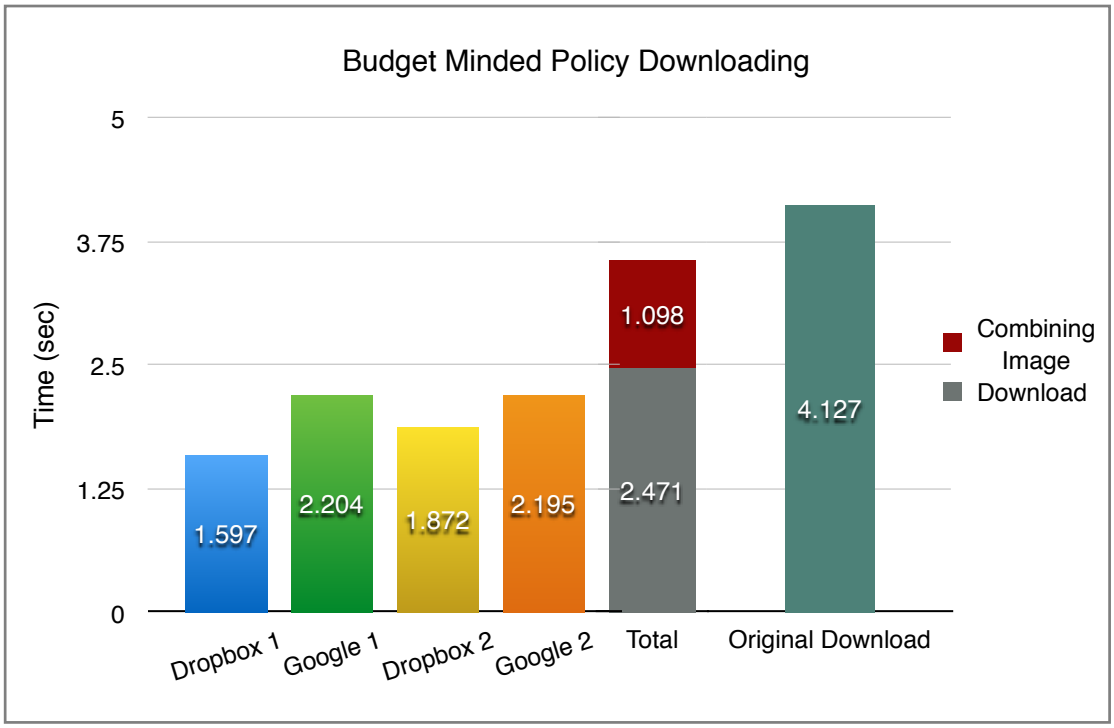

Figure 5.8: Budget Minded Policy Downloading

We see a linear decline in upload time based on file size. The total upload time is roughly half the time compared to our initial experiments. Combined with rescaling the image, we see a 18.74% decrease in upload time. This confirms that rescaling the image does indeed save storage space and improves performance. We see similar benefits in download time. It is expected that download time will improve simply due to the smaller file size. However, total time includes the combining of split files. Downloading and combining the image improves performance by 13.52% compared with initial experiments. The budget policy combined with the security policy proves to decrease storage space while improving performance.

5.3.4 Case 4 - Scaling File Size

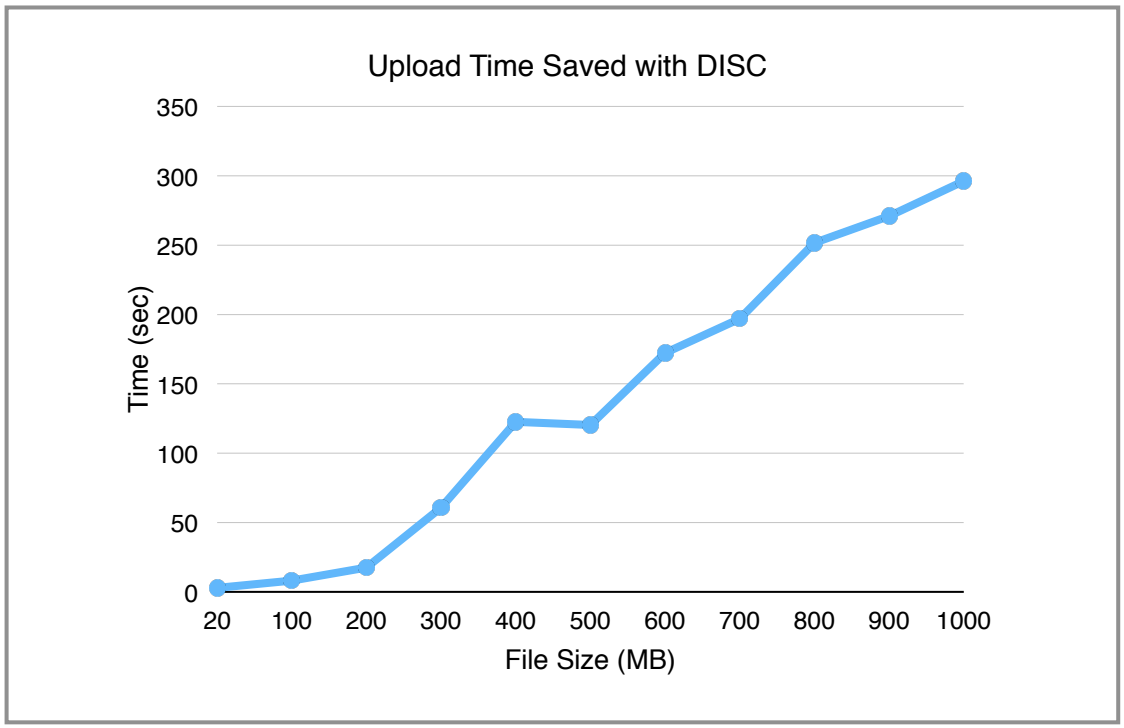

Figure 5.9: Upload Time Saved with DISC

To explore how well DISC scales with larger file sizes, we compare the traditional single provider approach with DISC and the security policy. Figure 5.9 shows the difference in upload time between using the single provider approach versus using DISC. We can note that, regardless of the file size, DISC will improve the upload

performance. File size and time saved have a positive linear correlation. DISC proves to save upwards of three-hundred seconds when uploading one-thousand megabytes of data.

5.3.5 Case 5 - Bandwidth Bottleneck

In this test experiment, we vary the number of cloud providers to determine the most ideal setting for the most optimal performance. Using Figure 5.10, we can see the affects of using too few cloud providers and too many cloud providers.

The time used to upload an image follows a parabolic pattern. Time continues to decrease until its minimum point at eight cloud providers. With each addition of a new cloud provider, each subsection of the original image decreases in size. The decrease in size improves the upload time for each cloud provider. After the minimum point, we begin to see an increase in upload time. Following the same logic as before, each cloud provider should take less time to upload. However, we run into a bandwidth issue. Even though each subsection has a smaller file size, the bandwidth and computer cannot handle the large number of open connections. The more connections open, the slower the performance is going to be. This is a common problem when dealing with bandwidth.

The time needed to download the same files just uploaded, follow a similar parabolic pattern. There is initially a steep slope, as seen in Figure 5.10, with fewer providers. This is caused by the poor performance of Dropbox. With each addition of a new provider, the file sizes decrease, allowing Dropbox to be comparable to Google Drive in download time. At the minimum point in the graph, we begin to see an increase in download time. With more connections open, the same problem occurs here as it did with uploading.

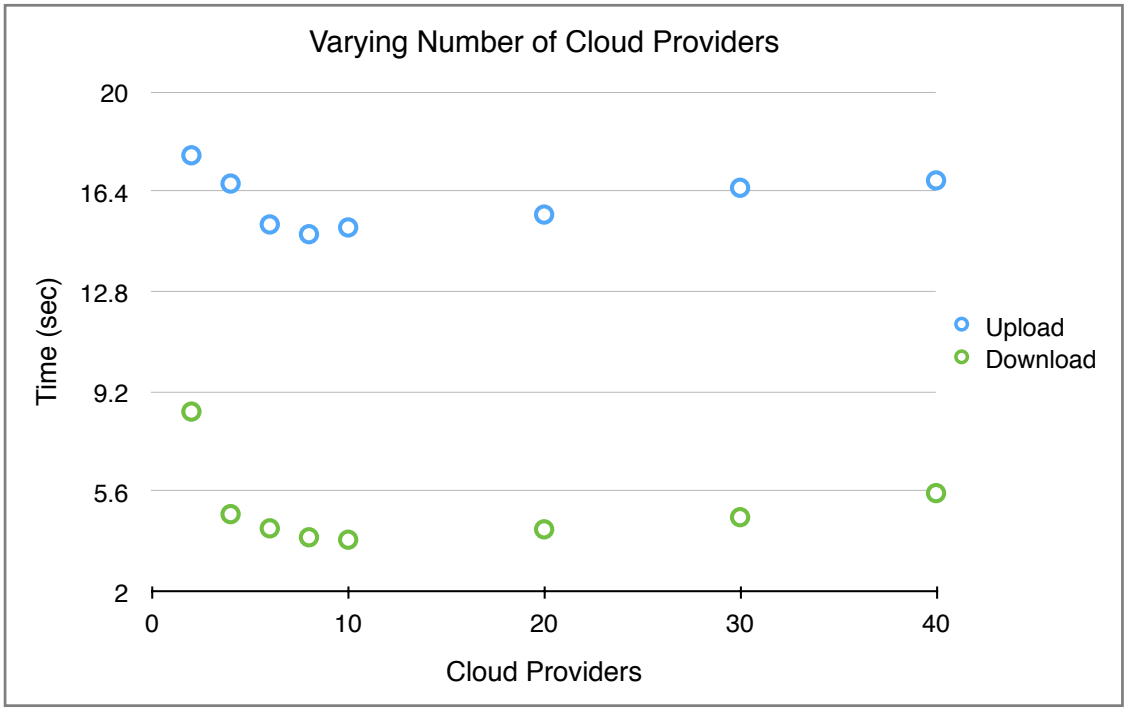

Figure 5.10: Varying Number of Cloud Providers

Overall, we see the same parabolic patterns from both upload and download. There is definitely a disadvantage using too few providers. Having too many providers causes worse performance, but still better than a single cloud provider. To achieve the best performance, we suggest to use six to ten providers.

5.3.6 Case 6 - Using a Manager

During this experiment, we run two tests using the security policy, with a Manager and without a Manager. We ran four iterations of the test to best illustrate how the Manager reacts to changing bandwidth performance. During each iteration with a Manager, we note the number of files processed per cloud provider and the total time to upload. With no Manager, there is no file migration so we only note the total time to upload. In Figure 5.11, the number of files processed is marked on the left and the total time to upload is marked on the right. Figure 5.12 illustrates the maximum and minimum time reduction achieved when using a Manager.

First, the Manager is able to skew the number of files processed based on the cloud provider's performance. Comparing all four iterations, there is no way of

predicting which cloud provider will process how many files. We see in Figure 5.11, how Google performs slow during one iteration, but performs well during a different iteration. These sudden changes are caught and corrected by the Manager.

Focusing on the two lines above the bar graphs, the total time to process all files is improved in all iterations. The varying times between using a Manager and without a Manager will vary unpredictably as well. On average, we were able to achieve 3.1 seconds decrease in total time compared to using no Manager. Imagine uploading one gigabyte of files using DISC with a Manager. Saving 3.1 seconds on 20.9MB results in at least of three minutes improvement per gigabyte. Incorporating a Manager into DISC shows how intelligently monitoring the data throughput and reacting to the results will improve performance significantly.

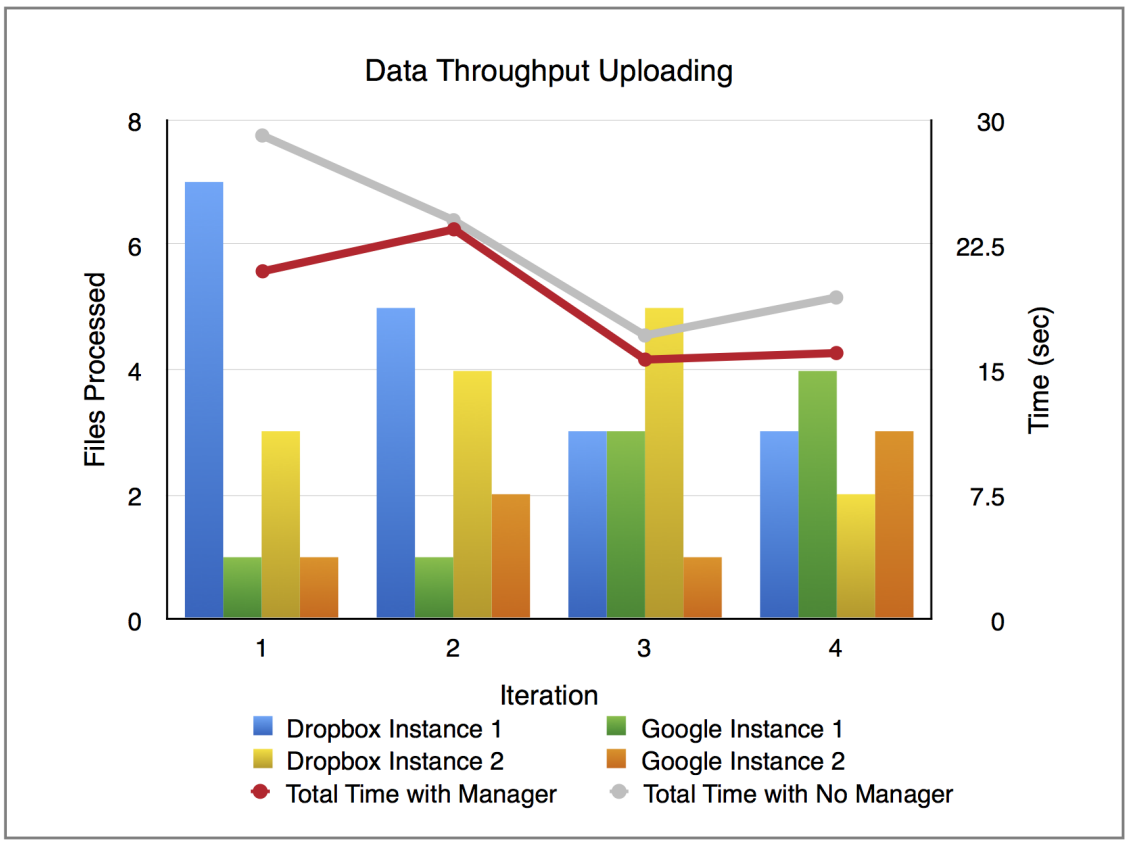

Figure 5.11: Data Throughput Uploading

Using DISC without a Manager allowed for a huge range of total processing times. One cloud provider has shown to finish over twice as fast as its counterpart.

The values in Figure 5.12 are calculated by taking the difference between the maximum and the minimum total processing times. The larger the difference, the more time between two cloud providers completing. This difference is considered wasted time. In an ideal scenario, zero seconds are wasted and all cloud providers require the same amount of time to complete processing the file. By using a Manager and migrating files between file queues, we can decrease the range in processing times resulting in less wasted time.

We use the same four iterations from before to illustrate the decrease in processing time. Three out of four cases have shown to save time with the addition of a Manager. Iteration three is a good example of unpredictable bandwidth. The reason for a Manager to perform slowly is due to a sudden increase or decrease in bandwidth during a separate test.

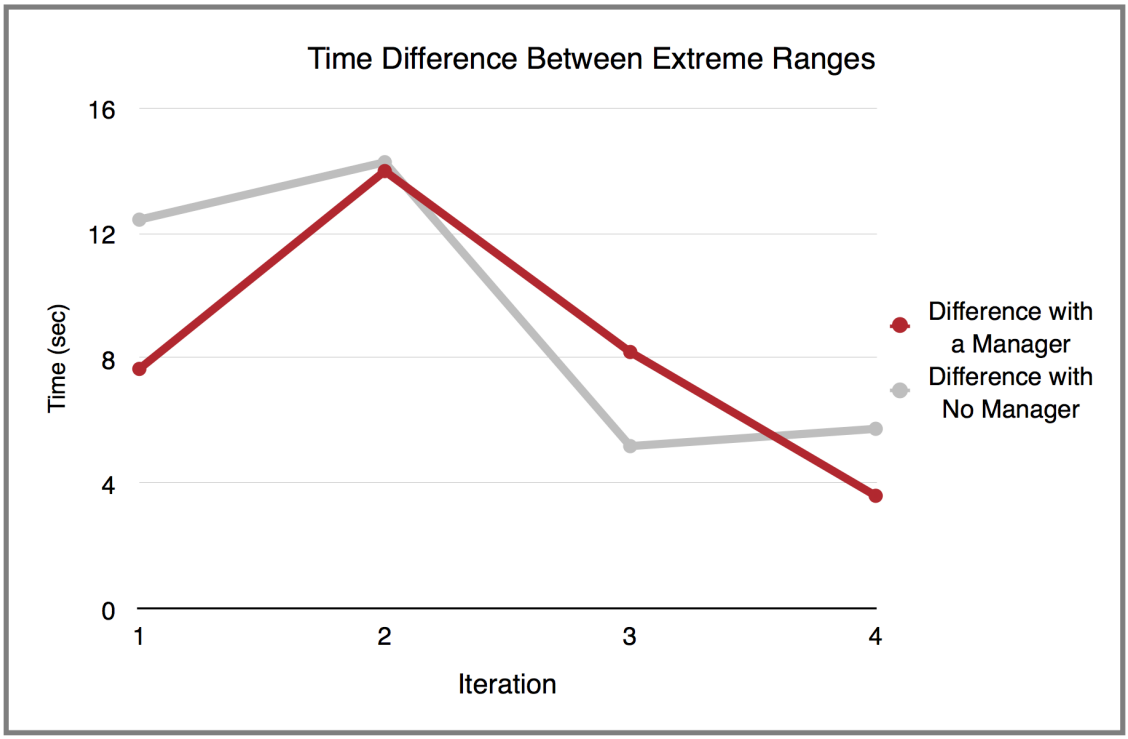

Figure 5.12: Time Difference Between Extreme Ranges

Using a Manager has shown to improve aspects for file processing that directly correlate to improved performance. The Manager migrates files between each cloud provider leveraging a load-balancing model. Load-balancing has shown on average a

14.28% decrease in processing time, equating to roughly three minutes per gigabyte. Most importantly, the Manager has shown to decrease the amount of wasted time by narrowing the gap between the slowest and quickest cloud providers.

6. RELATED WORK

6.1 RACS

H. Abu-Libdeh et al. [14] prove cases where a customer is susceptible to vendor lock-in. A lock-in is defined as being too costly to switch providers making them a hostage to its current provider. This problem can increase costs and possible data loss if their provider goes out of business. They propose a system known as *Redundant Array of Cloud Storage (RACS)*, a proxy to transparently redirect data across multiple storage providers. In tangent with the RACS system, they leverage a ZooKeeper to manage multiple RACS proxies.

While there are not any special alterations to the files being processed, they do use erasure coding to mimic a RAID system. Experiments have shown RACS to improve the time taken to retrieve and deliver files among RACS systems. The most CPU expensive operation is encoding and decoding erasure codes. They do not see this as a bottleneck initially until gigabit ethernet speeds can be achieved.

6.2 HAIL

K.Bowsers et al. [15] have worked on a system that takes a collection of servers and proves to a client that their file is intact and retrievable. They have introduced a system known as *HAIL: A High-Availability and Integrity Layer for Cloud Storage*. HAIL follows similar patterns with RAID, but HAIL manages file redundancy across cloud providers rather than local hard-drives.

HAIL uses a dispersal code to distributed each file block across n servers. File blocks are grouped together in a sequential order, but through *DISC*, we grouped file blocks using modulus as seen in our Security Policy. The dispersal code improves the redundancy of a file. HAIL embeds a *message-authentication code (MAC)* into the dispersal code. The MAC is only known to the client and is used to prove the files integrity.

Through experiments, HAIL illustrates significant improvements in ensuring availability and integrity. To achieve this, total processing times averages

around three-hundred seconds. Times increase linearly with each addition of a cloud provider. Over fifty-percent of this time is for encoding and decoding the files.

6.3 Hybrid Cloud Storage

The enterprise market is highly concerned with security and privacy, forcing them to look towards private cloud solutions. Kuo et al. [16] propose a hybrid cloud storage architecture, known as *Cloud Object Storage Appliance (COSA)*. COSA excels when installed on resource limited devices to combine both private and public cloud providers. The public cloud providers are used for backup of the private files. Encrypting the files is required before sending to the public providers.

There are benefits to using a hybrid cloud compared with either a single public or private provider. Storing the backup on public resources reduces the hard drive space necessary on private clouds enabling less costly maintenance. Additionally, COSA can provide high availability. Through their experiments, COSA is tested with a private cloud service with two-hundred employees. The application was able to serve its clients and handle enormous file objects.

7. DISCUSSION

7.1 Conclusion

Current approaches to cloud storage have shown to be inadequate, leveraging only a single provider. Security vulnerabilities, poor performance, and unpredictable reliability are some shortcomings to using a single provider. In this thesis, we have shown a new framework and model to solve these shortcomings.

Knowing cloud providers throttle the bandwidth, we can leverage the concurrent cloud storage model to take full advantage of the local user bandwidth by processing files concurrently. We have presented a middleware framework known as *Distributed Indexed Storage in the Cloud (DISC)* that leverages this model. Creating an abstracted storage class structure is important in decreasing the amount of work and time for developers to incorporate their own cloud service, whether public or private. Cloud storage is used by a very heterogeneous collection of devices. The choice of Python enables a wide range of devices to run this framework and leverage what DISC has to offer. As a result of uncontrollable network speeds, we introduce a Manager to tackle fluctuating bandwidth performance. Through the use of user policies, every application is written and configured to process given files based on the user preferences. During the experiments, we provide several test cases based on various scenarios that replicate how a user might define their preferences. We were able to strengthen security, improve reliability, and illustrate the benefits of a Manager. Using the various policies, we can dramatically reduce the total processing time upwards of eight seconds per twenty megabytes compared to single-provider approaches.

Cloud storage has been commonly sought after as a primary resource to store and share files. It is important to develop a user driven application through policies, while helping developers incorporate such a framework into currently existing applications. We expect DISC will change the way we interface with cloud providers.

7.2 Future Work

We hope this new framework is incorporated into developers own cloud storage applications. DISC has shown many significant enhancements to the cloud storage model. Initially, we want to expand the cloud storage options beyond Dropbox and Google Drive. Secondly, we want to take this framework to the mobile platform. When a user is on their smartphone, they use 3G/LTE networks which result in greater bandwidth fluctuations compared to WiFi. As described in this thesis, a Manager is an excellent solution to improve performance on an already limiting bandwidth. The User Policies illustrated throughout the thesis can ensure more security and availability for the users files when on the go. Lastly, there are many areas to improve how we upload video files. Video files can create enormous file sizes that require a significant amount of processing time. We want to research expanding our user policies to tackle this area of multimedia.

LITERATURE CITED

[1] "Why Cloud Storage is Growing in Popularity," Available: http://articles.bplans.com/why-cloud-storage-is-growing-in-use-and-popularity/ Accessed: Feb. 27, 2015.

[2] D. Slamanig, C. Hanser, "On Cloud Storage and the Cloud of Clouds Approach," *in Proc. of the 7th Int. Conf. for Internet Tech. and Secured Trans.*, 2012, pp. 649-655.

[3] Dropbox Inc., San Francisco, CA, USA, "Plans - Dropbox," Available: https://www.dropbox.com/plans/. Accessed: Dec. 29, 2014.

[4] Box Inc., Los Altos, CA, USA, "Plans and Pricing Box," Available: https://www.box.com/pricing/. Accessed: Dec. 29, 2014.

[5] Google Inc., Mountain View, CA, USA, "Buy and manage storage plans - Drive help," Available: https://support.google.com/drive/answer/2375123 Accessed: Dec. 29, 2014.

[6] Microsoft Inc., Redmond, WA, USA, "OneDrive Plans," Available: https://onedrive.live.com/about/en-us/plans/. Accessed: Dec. 29, 2014.

[7] "Number of pictures that can be stored on a memory device," Available: http://kb.sandisk.com/app/answers/detail/a_id/69/ Accessed: Dec. 20, 2014.

[8] "Number of hours of Hi-Def Video that can be stored on a memory device," Available: http://kb.sandisk.com/app/answers/detail/a_id/8407/ Accessed: Dec. 20, 2014.

[9] W. Kim, S. Kim, E. Lee, S. Lee, "Adoption Issues for Cloud Computing," *in Proc. of the 7th Int. Conf. on Advances in Mobile Computing and Multimedia*, 2009, pp. 2-5.

[10] W. Zeng, Y. Zhao, K. Ou, W. Song, "Research on Cloud Storage Architecture and Key Technologies," *in Proc. of the 2nd Int. Conf. on Interaction Sciences: Inform. Tech., Culture and Human*, 2009, pp. 1044-1048.

[11] G. Keizer. Network World Inc., Framingham, MA, USA, "Customers praise Microsofts no BS explanation of cloud service outage," Available: http://www.networkworld.com/article/2598886/cloud-computing/customers-praise-microsofts-no-bs-explanation-of-cloud-service-outage.html. Accessed: Dec. 29, 2014.

[12] "Image processing in Python," Available: http://scikit-image.org. Accessed: Dec. 30, 2014.

[13] R. Wolski, N. Spring, J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Comput. Syst.*, vol. 15, no. 5-6, pp. 757-768, Oct. 1999.

[14] H. Abu-Libdeh, L. Princehouse, H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," *in Proc. of the 1st ACM Symp. on Cloud Computing*, 2010, pp. 229-240.

[15] K. Bowers, A. Juels, A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," *in Proc. of the 16th ACM Conf. on Comput. and Commun. Security*, 2009, pp. 187-198.

[16] Y. Kuo, Y. Jeng, J. Chen, "A Hybrid Cloud Storage Architecture for Service Operational High Availability," *in Proc. of the 37th IEEE Conf. on Comput. Software and Applicat.*, 2013, pp. 487-492.