A Heterogeneous Hybrid Cloud Storage Service Using Storage Gateway with Transfer Acceleration and Diff Algorithm

Jamal Abdul Nasyir[1], Idris Winarno[2] and Udin Harun Al-Rasyid[3]
jamal.nasyir@gmail.com[1], idris@pens.ac.id[2], udinharun@pens.ac.id[3]
Polikteknik Elektronika Negeri Surabaya

Article Information

Submitted : 3 Aug 2022
Reviewed: 5 Aug 2022
Accepted : 30 Aug 2022

Keywords

Hybrid Storage, Storage Gateway, Transfer Acceleration, Intelligent Tiering, and Diff Algorithm

Abstract

Recently, the cloud service has the potential to replace conventional cluster and grid systems. The objective of migrating apps to the cloud is to minimize maintenance and procurement expenses while simultaneously boosting scalability and availability. However, embra=cing cloud technology created some challenges, such as the complexity of cloud storage. In addition, many clients underestimate if it is not plug-and-play. Each vendor has its access methods, and nonstandard application programming interfaces (APIs) make integrated applications, such as archiving or sharing data with cloud storage, complicated, costly, and require high throughput.Furthermore, organizations did not have many alternatives for implementing high-performance object storage systems in the cloud and on-premises data centers until now. In this paper, we would like to suggest a storage gateway as a solution to this issue and will optimize it using Transfer Acceleration and Diff algorithms to improve the performance, Intelligent Tiering to reduce costs, and Server-Side encryption for extra protection. Moreover, utilizing Storage Gateway has proven can provide more efficient integration between the on-premises data center environment and the AWS Cloud Storage ecosystem that is safer and more reliable. This technology can work in a common data center environment regardless of the vendor used by the company it can communicate seamlessly with the AWS Environment.

A. Introduction

The technology of using the cloud got the hype in this era due to the fast deployment, efficiencies, scalabilities, and big availabilities of networks asset that self. As a result, cloud computing has completely changed the game in the contemporary digital environment. Many businesses have moved their IT infrastructure there due to this technology. The cloud is undoubtedly a network of virtualized computers that are dynamically provided that performs parallel and distributive computing and are used by the user as a single computing resource in accordance with previously established "SLA" or Service Level Agreements. By enabling customers to access and store data remotely through cloud services, it would be possible to offer highly available and on-demand services without the hassle of managing local devices and software. [1] Cloud computing services have recently had a tremendous possibility to contribute as an alternative to traditional clusters and grid systems. Costs for acquisition and upkeep are decreased by moving our apps to the cloud. While increase=ed availabilities and scalabilities [2].

Although, there was some opposition when adopting cloud technology, such as integrating storage between premises and the cloud. However, cloud technology presents other challenges, such as merging on-premises and cloud storage. Moreover, some clients underestimated the suitability of this concept. As we already know, each brand has unique APIs and access to connecting apps. Consequently, it may not be plug-and-play [3]. Sadly, consumers do not currently have many options for deploying an object storage system between an on-premises data center and the cloud with high-performance capabilities. Based on that problem, we propose using the Storage Gateway technique to enable a secure and smooth integration between on-premises data center systems and collaborate it with the Diff Algorithm. We choose Xdelta as our Diff Algorithm to optimize our concept since it is a widely used de-facto standard implementation of diff algorithms of this generation. The main advantage of using a compression method can significantly reduce the delta file size. A data block will never appear in the delta more than once, it may be referred to more than once, but the block's actual data will only be loaded in the delta once. It is a significant difference from the previous approaches. This technology should be worked in almost all the data center environments regardless of the vendor it can communicate perfectly with the Environment of the Cloud.

B. Related Work

It has been several studies that have been concerned about the storage-gateway hybrid. An overview of cloud computing's data storage [1] A study review was conducted to discuss the common problems and issues in cloud storage and compared several storage gateways, primarily AWS Storage Gateway and IBM Storage Gateway.

Also, Lately, that was a study on Storage Gateway Medical Imaging on an Intelligent Cloud [4]. Communication delay is a serious problem that currently prevents the use of this concept in medical imaging applications.

Furthermore, HyCloud: Tweaking Hybrid Cloud Storage Services for Cost-Efficient Filesystem Hosting [5] has done some research on hybrid cloud storage has been done by hosting to optimize cost-effectiveness in AWS service.

Depend on the issue of adopting the technology of the cloud storage, which quite has big complexity and meets the need of users by using cloud storage to lower capital costs without lowering performance and productivity. In these segments, the authors Would like to present our initial attempt at a hybrid architecture that may link local data storage with online storage. The authors desire some systems that can optimize the transition from on-premises to the cloud while bridging the gap by utilizing CDN technology.

The data handling must also be upgraded to achieve greater control and data closeness from the storage gateway. By turning on those options, Bucket can automatically decide if a file is regularly viewed or not and determine which file has been moved to the deep archive category by classifying the entire data. Additionally, turn on the Bucket's Server-Side encryption for increased data security.

There are numerous well-known, significant public cloud providers, including AWS, Azure, GCP, Alibaba, Oracle Cloud, IBM Cloud, and many others. For this research, purpose author would like to use AWS providers to deploy our system design.

C. Preliminary Research

Previously we have done preliminary work related to this research [22]. We have successfully implemented the storage gateway that can work seamlessly to bridge the on-prem to the cloud and optimized that storage gateway using Transfer Acceleration from previous research. But, from that research, we also have some findings where the previous system uses a lot of bandwidth since they need to sync every single hour. That's why in this paper, the author would like to make some improvements that can help to reduce the bandwidth usage by implementing some simple algorithms called "Diff Algorithm." Typically, a diff displays the differences between two different file versions. Modern implementations also support binary files. The result is referred to as a patch or "diff.". By generating the diff file containing the part, the user has changed. Hopefully, it can reduce a lot of bandwidth, and we only need to upload the part that has been changed.

D. System Design

A. An Overview

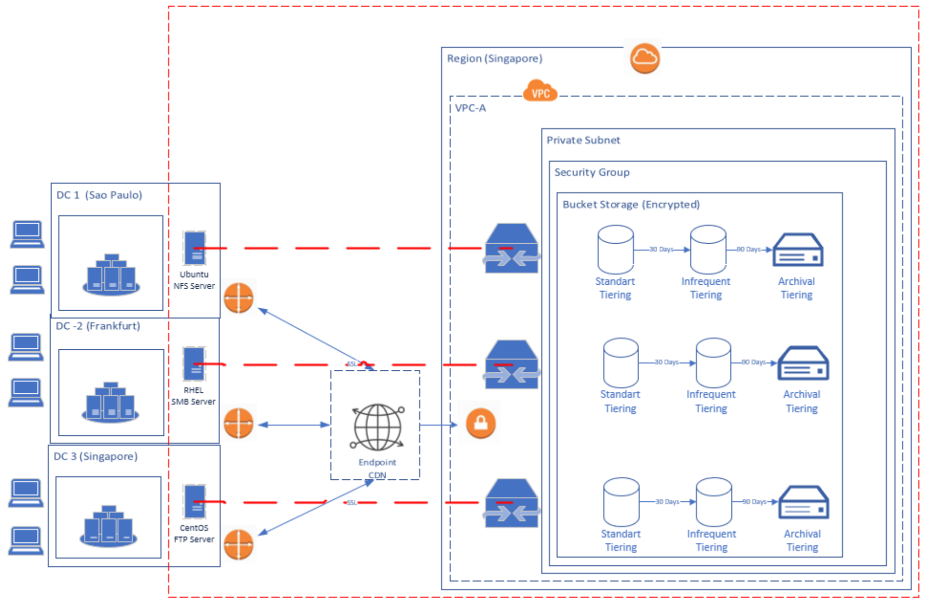


Figure 1. Proposed System Design

Fig 1 shows this Storage gateway works for heterogeneous, which could be used in several different OS environments, including RHEL, Ubuntu, and CentOS. Additionally, this Storage gateway can use in Windows environments. The connections will connect one to one with the bucket.

Integrating data between the branch and the headquarters will require a long route and extra time. Considering that many branches are spread out thousands of miles distant in a region. The authors intend to try to create some TA Technology to address this problem by employing this TA Technology, which will direct the data to the nearest edge point from the branch. By leveraging this link, the data transmission between the branch and headquarters is faster and more secure because the data transmission is not using public routes. The TA providers will use their global backbone to transport the data to the origin between their respective regions. Every single connection will be encrypted using SSL. In addition, using server-side encryption, all data on the server will be encrypted (SSE) using a Key Management System (KMS) to a Cryptographic key managed. It does not stop there, to enhance the system author would try to implement Delta Incremental data uploaded, which can detect data changes and find the differences between the old file and a new one if there are any changes to a specific file, and the system will only upload the delta data that has been changed and no need to upload entire data, The expected result of the technique will be reduced a lot of bandwidth usage. When the term "storage gateway" is used, it refers to a service that enables secure, the setting of cloud storage and the user's on-premises IT infrastructure is tightly integrated, as well as local storage devices or defined routes stored inside the server. Using this service, users could safely upload their data to the cloud for scalable and affordable storage. The storage protocols supported by this Storage Gateway are those utilized by our idea. Keeping utilized data on-premises and this storage gateway produces low latency performance and securely stores all encrypted user data copies in the bucket. Also, this technique may be used to disaster recovery as a cloud-hosted solution with a native computer system replicating the production environment [6].

Table 1. Specification Testing Environment

Environment	Name	Details
DC-1 (Sao Paulo)	CPU	2 vCPU
	Memory	4 Gb
	Disk	100 Gb
	OS	RHEL 8.1
DC-2 (Frankfurt)	CPU	2 vCPU
	Memory	4 Gb
	Disk	100 Gb
	OS	Ubuntu 20.04
DC-3 (Singapore)	CPU	2 vCPU
	Memory	4 Gb
	Disk	100 Gb
	OS	CentOS

As we can see on the TABLE I In this testing environment, we developed three distinct types of services running on three different OSs to keep prefixes and directories in sync with our storage gateway. While files are added to a bucket, intelligent tiering automatically classifies them based on their requirements, recursively copies new and updated files from the source directory to the destination directory.

B. Transfer Acceleration (TA)

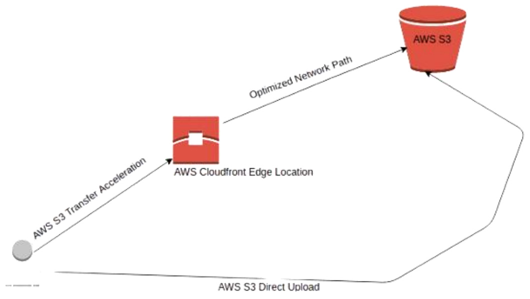


Figure 2. Transfer Acceleration Concept

Users can expedite the upload of information or material from around the globe to a single bucket by using transfer acceleration, often known as TA. Singapore can be used as a central bucket in this situation. As shown in Fig 2, by leveraging network protocol improvements, the provider's global backbone, and automatically sending data to the closest abuse edge location, Transfers may be sped up through transfer acceleration. Transfer acceleration makes bypassing a large portion of the public Internet and ISP bandwidth restrictions possible. Performance improvements often vary from 50% to 500%, depending on how close the bucket is to the intended application. The infrastructure required to support the services offered by TA providers is often housed at dozens or even hundreds of points of presence worldwide. When using a quick local internet connection, the user's application must upload large items or transmit data across continents. Our requests for foot objects must be directed to the endpoint to exploit this TA, and all requests may benefit from the transfer acceleration provided by the global provider backbone [7].

C. Data Handling

In this work, the author is trying to emphasize on two kind of data processing techniques that we want to implement. Server-side encryption for the bucket as a safeguard data from unwanted and illegal access while at rest (SSE-KMS). And Intelligent Tiering that can automatically handle our data to reduce bucket consumption costs.

1) Intelligent Tiering

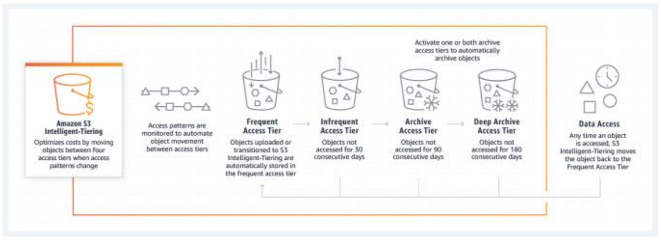

Amazon S3 Intelligent-Tiering
Optimizes costs by moving objects between four access tiers when access patterns change
Access patterns are monitored to automate object movement between access tiers
Frequent Access Tier
Objects uploaded or transitioned to S3 Intelligent-Tiering are automatically stored in the frequent access tier
Infrequent Access Tier
Objects not accessed for 30 consecutive days
Activate one or both archive access tiers to automatically archive objects
Archive Access Tier
Objects not accessed for 90 consecutive days
Deep Archive Access Tier
Objects not accessed for 180 consecutive days
Data Access
Any time an object is accessed, S3 Intelligent-Tiering moves the object back to the Frequent Access Tier

Figure 3. Data Handling Concept [7]

As we can see on the Fig 3 it showing there are 4 different tiers that we have on this storage bucket:

- Standard— This storage options are inexpensive and great for frequently accessed data
- Standard Infrequent — This storage is designed to the data with infrequent access and have standard-class pricing.
- Archive Storage — This classes are designed to archive valuable data for a long time with sparse access at a reasonable price. However, it will take some time to retrieve any data that the user may have saved here.
- Deep Archive Storage— This classes are designed to archive valuable data for a long time with sparse access at a reasonable price. However, it will take some time to retrieve any data that the user may have saved here.

Intelligent tiering is a storage type that may make storage more economical by shifting data to tiers that can be accessed at a lower cost. Intelligent-Tiering is designed to select the best storage class without sacrificing performance or adding operational overhead when irregular access patterns. Intelligent-Tiering may automatically save expenses by moving data at the granular object-level across access tiers as access patterns change.

Intelligent Tiering will keep an eye on the access patterns and automatically move items from one tier to the next without hiring a professional operator. For a monthly object monitoring and automation expense, Intelligent-Tiering will monitor the access patterns and move items from one tier to the next.

2) Protecting Data at Rest

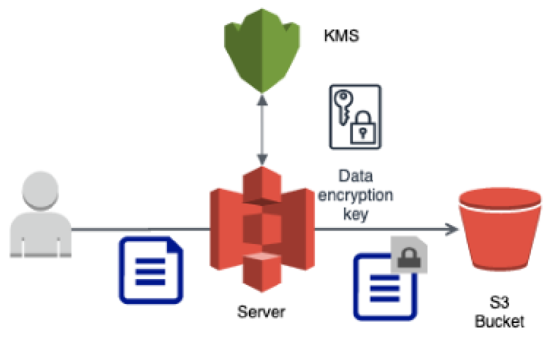


Figure 4. Server-Side Encryption Concept[7]

As shown in Fig. 4, for server-side encryption, a unique encryption key will be generated for each individual object. The data will be encrypted using the Advanced Encryption Standard (AES) and a 256-bit key (AES 256). The master key will be kept secret and periodically rotated to protect the encryption key.

Moreover, the authors also use the SSL protocol to protect the data while it is in transit, securing it twice.

D. Diff Algorithm

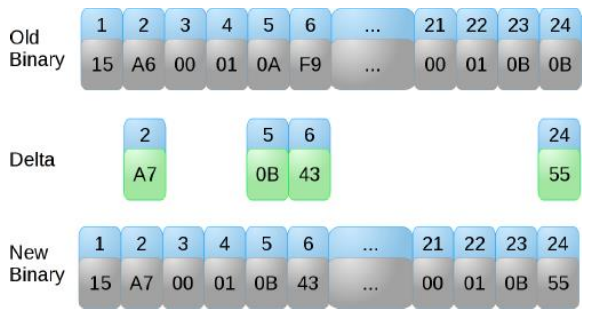


Figure 5. Delta Data Algorithm Simple Concept

To understand more about the Diff Algorithm, kindly refer to Fig 5 above to understand how this delta data can work in the byte data example. Delta data incremental upload, or what we can call as Diff Algorithm, was only a nice way to say that we need to upload files that change or new files. Instead of uploading every single piece of data to the bucket each time, the system just uploaded the newly created or modified file. Additionally, this will save their bandwidth.

Diff Algorithm is an alternative term for storing or sending data as differences (deltas) between sequential data and the whole file. Delta encoding is widely known as delta compression when an updated archive history is needed (e.g., in revision control software). The changes are stored in separate files known as "deltas" or "diffs." Delta encoding considerably minimizes data redundancy when only minor changes are made, such as modifying a few words in a vast manuscript or a few entries in a large database. Collections of unique deltas use less space than their unencoded equivalents.

By trying to implement this delta data, our goal is to minimize the data usage when synchronizing all data for this storage gateway. Because there are many of data in our storage and we cannot upload the whole data every hour, we need a solution that can save our bandwidth without reducing our RPO (Recovery Point Objective) and increase performance. Therefore, for any application, selecting the optimum delta algorithm requires reliable, comparative data on each technique's effectiveness.

There are several forms of redundant data in storage systems, and a variety of data reduction strategies are available to handle them. Please refer to TABLE II for the specifics:

Table 2. Data Reduction Approaches Comparison

	General Compression	**Delta Compression**	**Data Deduplication**
Objects	All Data	Similar Data	Duplicate Data
Granularity	String / Byte-Level	String-/Byte-level	Chunk-level
Rep Techniques	Huffman coding / Dictionary coding	Copy/Insert-based Delta encoding	CDC & Secure signature
Approach Dates	1970s	1990s	2000s
Rep. Prototypes	GZIP, Zlib , Zstd	Xdelta, Zdelta, Ddelta	Venti, LBFS, DDFS

- General compression (i.e., traditional lossless compression) approaches, such as GZIP, focus on the internal compression of files or data chunks and reduce byte- and bit-level redundant material by using Dictionary coding and Huffman coding. Due to the subtle redundancy removal approaches, general compression methods are usually time-consuming and only compress data over a small region (e.g., a single file or a chunk).
- Delta compression aims to eliminate redundancy between similar files/chunks. In general, it (e.g., Xdelta) finds repeated strings using the Rabin-Karp string matching technique and then encodes matched (duplicate) strings with the "Copy" instructions and the strings that do not match the "Insert" instructions. Furthermore, Zdelta compresses mismatched strings at the byte level using Huffman coding.
- Data deduplication is a chunk-level compression strategy for large-scale storage systems that eliminates redundancy in fine details quickly. This typically breaks files into roughly equal-length chunks using Content-Defined Chunking (CDC) and then calculates each chunk's secure signature (or termed fingerprint, e.g., SHA1) for duplicate matches. If two chunks have the same secure signature, they are duplicates. This approach is simple, fast, and easy to implement in large-scale storage systems. However, there is still a lot of redundant data between the extremely similar but non-duplicate chunks/files, which this thing can be handled via delta compression.

In contrast to general compression and data deduplication, delta compression may remove redundancy among files/chunks that are quite identical, as was stated above. As a result, it has drawn more attention in recent years. Implement delta compression on top of deduplication to further reduce redundancy and speed up WAN replication of backup information. This shows that delta compression may assist achieve a compression ratio around 2X greater than possible using conventional methods. General compression (i.e., traditional lossless compression) approaches, such as GZIP, focus on the internal compression of files or data chunks and reduce byte- and bit-level redundant material using Dictionary coding and Huffman coding. Due to the subtle redundancy removal approaches, general compression methods are usually time-consuming and only compress data over a small region (e.g., a single file or a chunk).

Table 3. Delta Algorithm Comparison

Algorithm	**Chunking**	**Hashing**	**Indexing**
Myers Algorithm	Rabins	SHA-256	Arbitrary indices
BSDiff	Larsson & Sadakane's qsufsort	SHA1	Suffix Sort
Open-vcdiff	Windows	BlockHash	suffix trees
Gdelta	FSC	Gear	Array
Zdelta	FSC	Adler32	Hash Table
Ddelta / Edelta	CDC	Spooky/ xxHash	Hash Table
Xdelta	FSC	Adler32	Array

Note:
1. The term "chunking" refers to combining smaller bits of information into bigger ones.
2. FSC refers to using a byte-wise sliding window to roll across chunks of material to create fixed-sized and overlapping words and optimize duplicate-word matching.
3. The Zdelta hash table determines the best match during indexing.
4. CDC refers to breaking the chunks into multiple variable-sized and non-overlapping words to match duplicate words easily.
5.'windows' chunks are some processes VCDiff uses to deal with memory limitations to partition the input file into chunks and process them separately.

In the end, we chose xDelta for our research because it has a high-quality native code implementation with O(n). complexity. In other words, in the worst-case scenario, the algorithm discards a larger delta than the original message, but we don't waste much time generating these delta. This allows us to easily handle the trade-off between the bandwidth saved by generating deltas and the CPU expenses associated with generating these deltas.

E. Result and Discussion

Identical to what the authors said in section III (System Design) and Fig. 1, this storage gateway operates flawlessly with three independent services, including NFS, SMB, and FTP. All data will be automatically encrypted by the Default Encryption, which was activated upon uploading the data into a bucket using the cryptography key we previously established with an asymmetric key. This encryption and decryption process will require a key pair.

E. Storage Gateway Benchmarking

Based on our tests, our system can sync and replicate a whole data set and bucket in our test environment without any difficulties after placing our script in the crontab to guarantee that sync replication would occur once per hour. In addition, we tried to add a new file or folder; the system will upload this new item to the bucket according to the following schedule.

Utilizing the AWS Command Line Interface, the applications are linked to AWS cloud storage using this storage gateway. It simplifies and reduces infrastructure in the data center and remote sites. Even though the service provides cloud storage that connects with your existing applications, the storage gateway utilizes industry-standard storage protocols to provide a smooth administration experience for the user. The file uploaded to an NFS, SMB, or FTP server would be replicated using an AWS CLI command and kept in the bucket as a durable object, allowing users to use it with other AWS services, such as analytics, and machine learning, and lambda functions.

F. Transfer Acceleration (TA)

Table 5. Write Time Test Result

Site	Run	With TA (s)	Without TA (s)	Efficiency (%)	Min	Max	Std Dev	Average (%)
DC-01 (Sao Paulo)	1	27,462	38,148	28,0	25,5	29,6	1,7	27,70
	2	26,394	35,427	25,5				
	3	26,619	37,813	29,6				
DC-02 (Frankfurt)	1	20,027	29,713	32,6	22,4	32,6	4,5	26,30
	2	22,461	28,931	22,4				
	3	20,612	27,096	23,9				
DC-03 (Singapore)	1	17,646	18,209	3,1	0,9	3,1	0,9	1,97
	2	17,436	17,595	0,9				
	3	17,75	18,096	1,9				

Table 6. Latency Test Result

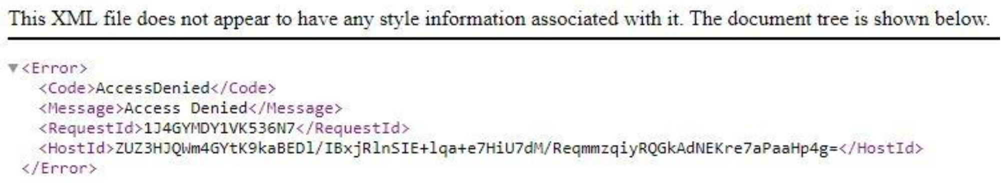
Site	Run	With TA (ms)	Without TA (ms)	Efficiency (%)	Min	Max	Std Dev	Average (%)
DC-01 (Sao Paulo)	1	0,37	334	99,9	99,8	99,9	0,0	99,87
	2	0,51	333	99,8				
	3	0,39	328	99,9				
DC-02 (Frankfurt)	1	0,969	155	99,4	99,3	99,4	0,0	99,34
	2	1,03	154	99,3				
	3	1,04	154	99,3				
DC-03 (Singapore)	1	0,514	0,932	44,8	14,6	44,8	13,2	26,51
	2	0,473	0,592	20,1				
	3	0,422	0,494	14,6				

The authors have attempted to conduct the test in this section three-time write benchmark to do a write benchmark test three times to compare TA on and off and determine the extent to which the TA can alter the data transfer. To demonstrate whether we use type 2 (two) benchmarking methodologies to accelerate the data transfer using this Transfer Acceleration

1) Write Time Test - Making a random file with a 1 GB file size is the first method of measuring the write time test (Gigabyte). To generate this arbitrary file, we utilize the "dd" tool. Then, using Linux's time function, we must determine how long it will take to upload the 1GB file to the cloud. And as a result, Transfer Acceleration functions quite effectively. And based on TABLE IV we can conclude that The smooth operation and high throughput of this storage gateway will be very beneficial for transporting bigger things over longer distances
2) Latency Test - Our second method for determining the latency test between on-premises and the cloud uses the Ping command with the "U" argument to trace the round trip time of the package. According to the table, the average efficiency rate of delay between DC-01 (Sao Paulo) and DC-02 (Frankfurt) is rather high at a rate of more than 90%. The test was attempted three times at each site by the authors.

Considering the results of the tests in this section, this TA has proven to be able to enhance several network connections between the bucket and the premises. If a higher throughput is required, Transfer Acceleration is the preferred option. The TCP connection is optimized, and more intelligence is provided between the client and the bucket.

G. *Server-Side Encryption*



This XML file does not appear to have any style information associated with it. The document tree is shown below.

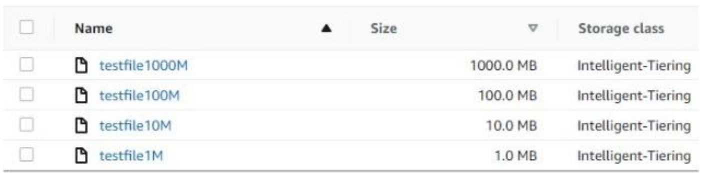
```
▼<Error>
   <Code>AccessDenied</Code>
   <Message>Access Denied</Message>
   <RequestId>1J4GYMDY1VK536N7</RequestId>
   <HostId>ZUZ3HJQWm4GYtK9kaBEDl/IBxjRlnSIE+lqa+e7HiU7dM/ReqmmzqiyRQGkAdNEKre7aPaaHp4g=</HostId>
 </Error>
```

Figure 6. Server-Side Encryption Result

The system will show an error because it could not decrypt the file when unauthorized access was attempted. The bucket will be able to encrypt the file at the bucket level rather than the object level, securing user data from hackers or unauthorized people if Server-Side Encryption was successfully utilized to encrypt the file and install it on the bucket using a certain key management system (SSE-KMS). According to the Server-Side Encryption (SSE) compliance requirement, the user must transmit encryption information with each request for object storage to encrypt all objects stored in a bucket without utilizing the default encryption method.

According to the test results in Fig. 6, server-side encryption operates as anticipated. However, only the IAM user indicated who has access to the bucket may decrypt the file.

H. Intelligent Tiering



Name	Size	Storage class
testfile1000M	1000.0 MB	Intelligent-Tiering
testfile100M	100.0 MB	Intelligent-Tiering
testfile10M	10.0 MB	Intelligent-Tiering
testfile1M	1.0 MB	Intelligent-Tiering

Figure 7. Intelligent Tiering Show Result

The author attempts to upload our sample data in this section, and each files placed on the bucket that would be assigned automatically by intelligent tiering as we can see the result on the Fig 7.

This tiering might be useful if the user is unclear about the object's frequency of access. Customers would like this sort of Tiered Storage. This functionality may be activated when new system items are created. Using life-cycle policies, the Customers may also shift the object manually. Unlike intelligent tiering, which can run automatically, this life cycle policy will need more effort from the user, since each policy must be set up and stated manually.

If customers are certain that their items will be consumed infrequently, the Standard-IA storage class is recommended for the greatest cloud cost reductions. Customers may enhance cloud savings and reallocate their time by letting the bucket to choose the optimal tiers for each item and move them as needed. This is undoubtedly the most helpful feature for customers.

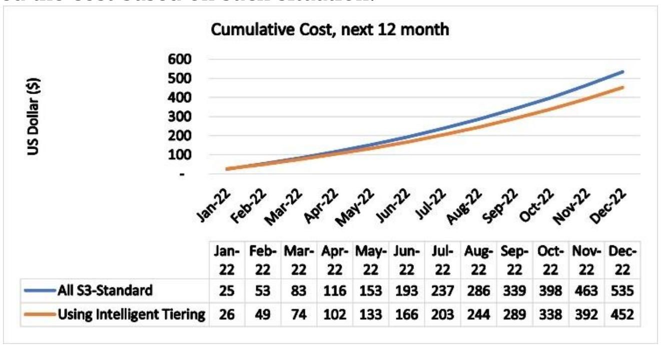
I. Cost Efficiency

As we mentioned before, implementing intelligent tiering aims to reduce storage costs by automatically transferring data to the most cost-effective access tier without impacting performance or adding operational burden. This part evaluates and calculates various cost estimates for all of the data in the bucket using both traditional and intelligent tiering.

For the cost optimization analysis, the author makes several scenarios to find out how much efficient the storage gateway in question:

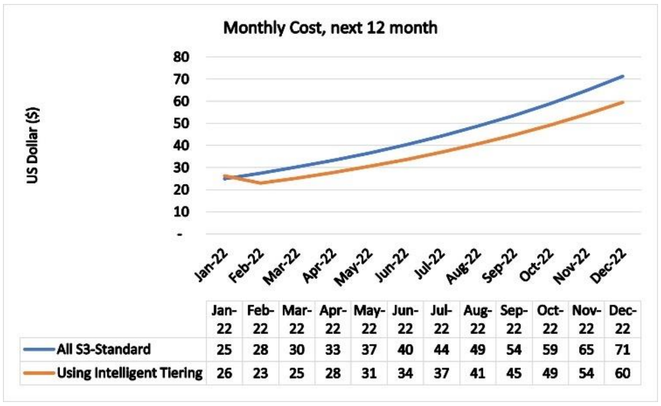
- 1 terabyte of storage containing 1 hundred thousand objects is hosted by the author.
- The amount of "hot" data (often accessed data), or the average object size, is 10MB.
- Monthly storage growth is anticipated to be 10%.

We have approximated the cost based on such situation.



	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
All S3-Standard	25	53	83	116	153	193	237	286	339	398	463	535
Using Intelligent Tiering	26	49	74	102	133	166	203	244	289	338	392	452

Figure 8. Monthly Cost for next 12 Month



	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
All S3-Standard	25	28	30	33	37	40	44	49	54	59	65	71
Using Intelligent Tiering	26	23	25	28	31	34	37	41	45	49	54	60

Figure 9. Cumulative cost for 12 months

As shown in Fig 8 and Fig 9, all of our objects are in the tier bucket Standard. We only give a broad review the benefits of intelligent tiering. The cost is roughly 1.8 X lower per GB when intelligent -tiering promotes an object to the Intelligent-Infrequent tier. Naturally, not all the items will be shifted to an uncommon tier. This tier only contains items that have been inactive for 30 days. Additionally, Glacier or Glacier Deep Archive, cost 5.8X and 20X less per GB than S3-Standard, respectively, are not considered in this comparison.

We can observe from that graph that first-month costs with the new intelligent tiering are higher depending on our amount of objects because of the one-time object move fees. Intelligent Tiering often generates a return within 1-2 months. Additionally, with our condition, the 12-month cumulative The Intelligent Tiering results in savings of $82.89.

J. Delta Data (Diff Algorithm)

In this section, authors try to measure how effective Delta Data is in reducing file size and bandwidth usage. To run this Xdelta script we use Linux's crontab feature which will run every hour we just need to put that script inside the crontab, so it will generate a new delta data from our existing file and the storage gateway script will handle the rest of it to sync up to the cloud.

Identifying an appropriate data set for statistical significance and appropriate real-world applications was the first challenge we faced while attempting to develop a delta benchmark. The most crucial characteristic of any benchmark for delta algorithms is the presence of several instances of change. This implies that both the magnitude of the changes reflected, and the files' size must vary substantially. Large modifications to small files and minor modifications to large files must be included, as well as minor modifications to small files and large modifications to large files.

In addition, the benchmark should include a variety of formats, such as pure text, pure object code, and pseudo text because Word processors create pseudo text, a string of text with binary data inserted throughout, and rare line breaks. As word computers, spreadsheets, and multimedia data grow more widespread, pseudo text and binary forms are becoming more significant [16].

These types of data sets are present in GNU software. The Free Software Foundation is responsible for making numerous software projects available in updated forms. The test suite that the authors used consisted of two versions of

GNU GCC, versions 9.4.0 and 9.5.0, as well as two versions of GNU Emacs, versions 28.1 and 27.1. With this version, any given file can have a large range of variations from one revision to the next. The sizes of the files range from 0 to 200KB.

Table 6. Delta Algorithm Test Result

	Size	**Files Count**	**Size**	**Files Count**		
GCC	548421211	94646	548761899	94878	124531017	77.3%
Emacs	243673061	6022	259158350	6587	70144984	72.9%

Please refer to TABLE VI above to understand the result we obtained from our test. As we have mentioned earlier, for this benchmarking, we try to provide two different files, GCC & Emacs then we create the Delta from those two different versions. From our test, we were able to save a lot of bandwidth up to 77%+ per our test by only uploading a delta file instead of us needing to upload every latest file.

F. Conclusion

This Storage Gateway can be one of the solutions to Hybrid Cloud Storage. Furthermore, utilizing the storage gateway with all the techniques we mentioned before can be more secure and efficient and have seamless integration.

From the efficiency perspective, implementing the Diff algorithm into our Storage Gateway can significantly reduce the file size. Since that algorithm can compute the differences between two versions of a file and do some compression on the diff file that only saves the part that has been changed, this diff algorithm was proven can reduce a lot of our bandwidth up to 77%. Furthermore, once we have the patch file from the diff algorithm, transfer acceleration will take over the part and optimize the latency by finding the best route from the source to the destination using an optimized path rather than a public route. This TA will handle the data carefully, and this TA has up to 99% efficiency on the Latency Test and up to 27% on the Write time test.

Then, from the security and cost perspective of implementing SSE, it can be an additional safeguard to protect our data confidentiality when placed on the cloud. And for cost efficiency, intelligent tiering successfully reduce the cost without more effort to change the tiering manually. From our test scenario, this IA can reduce the cost up to $82.89

For future works, authors will try to find the best combination of Xdelta with another data reduction approach to add more efficiency to this concept.

G. References

[1] I. Odun-Ayo, O. Ajayi, B. Akanle, and R. Ahuja, "An overview of data storage in cloud computing," in *Proceedings - 2017 International Conference on Next Generation Computing and Information Systems, ICNGCIS 2017*, Nov. 2018, pp. 38–42. doi: 10.1109/ICNGCIS.2017.9.

[2] E. Rolloff, M. Diener, A. Carissimi, and P. Navaux, "High Performance Computing in the cloud: Deployment, performance andcost efficiency," Mar. 2012. doi: 10.1109/CBD.2015.40.

[3] A. Paul Rajan and S. Shanmugapriyaa, "Evolution of Cloud Storage as Cloud Computing Infrastructure Service," *OSR Journal of Computer Engineering (IOSRJCE)*, 2012, doi: 10.1109/CEC.2011.45.

[4] C. Viana-Ferreira, A. Guerra, J. F. Silva, S. Matos, and C. Costa, "An Intelligent Cloud Storage Gateway for Medical Imaging," Jul. 2017, [Online]. Available: http://arxiv.org/abs/1708.06334

[5] Y. Cui, M. Ruan, Z. Li, and E. Zhai, "HyCloud: Tweaking Hybrid Cloud Storage Services for Cost-Efficient Filesystem Hosting," *IEEE/ACM Transactions on Networking*, 2020, [Online]. Available: https://github.com/iHyCloud/hycloud-demo.

[6] J. Baron and S. Kotecha, "Amazon Web Services-AWS Storage Options Storage Options in the AWS Cloud," 2013. [Online]. Available: http://aws.amazon.com/whitepapers/

[7] AWS, "Amazon Simple Storage Service User Guide." 2019.

[8] C. M. Ruth Paul, "Experimenting with AWS Direct Connect using Chameleon, ExoGENI, and Internet2 Cloud Connect," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, Jan. 2019, pp. 1–651. doi: 10.1007/978-3-319-33769-2.

[9] R. Sieger, S. Grob, and A. Schill, "SecCSIE: A Secure Cloud Storage Integrator forEnterprises," *2011 IEEE Conference on Commerce and Enterprise Computing978-0-7695-4535-6/11 $26.00 © 2011 IEEEDOI 10.1109/CEC.2011.45252*, 2012.

[10] J. Wang, L. Yang, H. Zhang, Z. Xu, and Y. Guo, "PROXY: A high cost-performance cloud storagegateway," 2015. doi: 10.1109/CloudCom.2012.6427549.

[11] I. Saeed, S. Baras, and H. Hajjdjab, "Security and Privacy of AWS S3 and Azure Blob Storage Services," 2006.

[12] V. Persico, A. Montieri, and A. Pescape, "On the Network Performance of Amazon S3 Cloud-Storage Service," in *Proceedings - 2016 5th IEEE International Conference on Cloud Networking, CloudNet 2016*, Dec. 2016, pp. 113–118. doi: 10.1109/CloudNet.2016.16.

[13] Liu Shenling, Zhang Chunyung, and Chen Yujiao, "HASG: Security and Efficient Frame for Accessing Cloud Storage," *China Communication*, 2018.

[14] C. Gopinaath and C. Kiruthika, "A server side encryption for cloud storage with federation sharing in hybrid cloud environment," in *Proceedings - 2017 International Conference on Technical Advancements in Computers and Communication, ICTACC 2017*, Oct. 2017, vol. 2017-October, pp. 128–131. doi: 10.1109/ICTACC.2017.41.

[15] Q. Zhang *et al.*, “DeltaCFS: Boosting Delta Sync for Cloud Storage Services by Learning from NFS,” 2017.

[16] J. J. Hunt and W. F. Tichy, “Delta Algorithms: An Empirical Analysis,” *ACM Transactions on Software Engineering and Methodology*, vol. 7, no. 2, 1998.

[17] J. M. Uc, B. J. Mogul, H.-P. Company, and K. Vo, “The VCDIFF generic differencing and compression data format,” 2002.

[18] Wang Y, DeWitt DJ, and Cai JY, “X-Diff: An Effective Change Detection Algorithm for XML Documents,” 2003.

[19] Y. S. Nugroho, H. Hata, and K. Matsumoto, “How different are different diff algorithms in Git?,” *Empirical Software Engineering*, vol. 25, no. 1, pp. 790–823, Jan. 2020, doi: 10.1007/s10664-019-09772-z.

[20] A. Tridgell and P. Mackerras, “The rsync algorithm,” 1996. [Online]. Available: http://cs.anu.edu.au/techreports/

[21] Dieter Jonathan, “ON BINARY DELTA ALGORITHMS,” 2009. https://www.jdieter.net/posts/2009/11/06/on-binary-delta-algorithms/ (accessed Jun. 24, 2022).

[22] J. A. Nasyir, I. Winarno, and M. U. H. al Rasyid, “A Heterogeneous Hybrid Cloud Storage Service Using Storage Gateway with Transfer Acceleration,” in *International Electronics Symposium 2021: Wireless Technologies and Intelligent Systems for Better Human Lives, IES 2021 - Proceedings*, 2021, pp. 185–189.