ASTES

*Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 2, 57-63 (2020)*

*www.astesj.com*

*Special Issue on Multidisciplinary Sciences and Engineering*

**ASTESJ**

**ISSN: 2415-6698**

# Performance Evaluation and Examination of Data Transfer: A Review of Parallel Migration in Cloud Storage

Mimouna Alkhonaini*, Hoda El-Sayed

*Department of Computer Science, Bowie State University, 20715, USA*

A R T I C L E I N F O

*Article history:*
*Received: 05 November, 2019*
*Accepted: 21 February, 2020*
*Online: 09 March, 2020*

*Keywords:*
*Parallel*
*Transfer*
*Cloud*

A B S T R A C T

*Recent years have seen a continued pattern of development in the cloud computing field. Numerous approaches to maximize file transfer capacity are still completely standing for use on cloud computing storage; however, they do not maximize the advantage of data migration scalability and elasticity in cloud storage. One potential problem is that elasticity takes time; however, the scalability attributes that have not been fully exploited include multicore chips and parallelization that can further be leveraged to enhance the overall data transfer performance and efficiency. In that regard, considerable effort has been directed to multiprocessors. Such systems involve a plurality of processors or functioning units capable of independent operation to process separate tasks in parallel. Nevertheless, the penalization is complicated when a task requires several resources or signals to proceed with meaningful computation. Thus, accommodating equitable priority among tasks further complicates operations. In this paper, we propose a parallel server to cloud storage transfer system in which parallelism method can only be utilized in case of transferring a large number of files and applied in order to increase the transfer throughput. The data is transmitted into several chunks via TCP network within the same period slot in a single data path which indicates dataflow on parallelism. Our target in this system is that increasing number of processors and the problem size will simultaneously maintain the efficiency of the data transfer system. The proposed model is based on the combination of dynamic segmentation, CRS, AES, and hashing. In summary, the proposed model shows the potential to enhance the performance by increasing the data transferability. The performance of the proposed model will be measured with the help of comparing the average execution time with the number of processors and speedup of the entire parallel system.*

## 1. Introduction

The cloud computing involves the provision of on-demand computing services to the customers. These services are similar to the usual services that can be obtained by using the physical computing equipment such as storage devices, servers, networking and many more. As the realization of the benefits of cloud computing come to light, many businesses, companies, organizations, and in stitutions are switching to the cloud services. Cloud services are provided by the cloud service providers (CSP) who are responsible for manning all the infrastructures, software, and the platforms to which the customers subscribe. Performance remains a significant concern for the consumers who are still struggling with the decision of moving to the cloud. Though there are many significant standards in cloud computing, the primary standard is on the data migration performance from the non-cloud infrastructure to the cloud. This paper is an extension of the work which was initially published and presented in IEEE 20th International Conference on High-Performance Computing and Communications [1].

*Mimouna Alkhonaini, COSC Department, 14000 Jericho Park Road, Bowie, MD 20715, 301-860-3964 & alkhonainim0522@student.bowiestate.edu

Typically, a multiprocessor includes a variety of computational units, a memory, a control, and at least one I/O processor. The functions of a job are initiated and processed, sometimes being moved from one computational unit to another. In the course of such operations, tasks must be synchronized and, in this regard, a function may require data from another function supplied through the I/O processor [2]. Under such circumstances, there is an issue in increasing number of processors against problem size.

The features of parallelism have been known since Babbage's effort to construct a mechanistic computer [3]. Industrial and academic researchers have studied every imaginable aspect of parallel computation. A system becomes slower due to combined cumbersome computation processes, therefore, there is a need of using parallel computing to improve and provide high levels of execution performance efficiency. Moreover, this research also provides the design and validation of the analytic performance of the parallel migration to enhance the speed of the system.

Related studies in the fields of data migration via parallelism have gained focus over the past few years. However, most of the proposed studies only get the standpoint of either a cloud provider or cloud consumers, not both. In the present work, the proposed scheme shall concentrate on allowing the cloud provider and also the consumer to accomplish data migration efficiently by exploiting potential improvements found in consumer PC multicore chips. Moreover, future cloud storage environments will be primarily designed and built on top of multicore technology.

In this paper, we propose a parallel migration system that can be made cost-optimal by adjusting the number of processors and the problem size. The parallel migration is achieved by dynamic distribution for the combination of file slicing, cryptography, fault tolerance, and hashing.

The remaining paper is structured in a manner that section 2 presents the related works in this field. Section 3 defines and models how the proposed system achieves parallel migration by dynamic distribution. Section 4 shows the experimental outcome and section 5 concludes the paper with some future remarks.

## 2. Related Works

In this section, we discuss some of the approaches related to the data migration parallelism as well as increasing file transfer bandwidth.

HTTP, SFTP, and SCP are the standard data migration communication protocols for point-to-point transfer. In terms of elasticity, the improvement is based on the work of [4] that helps to scale down the repetitive data migration through developing "checksum comparison" and the work presented in [5] prevents resource conflict by distinguishing "back-pressure". However, still, these approaches have thus far to capitalize on cloud elasticity. In our method, we aim at matching the amount of resources allocated to data transfer in parallel mode with the amount of resources it actually requires, avoiding over-provisioning or under-provisioning.

In terms of connection speed, the existing odd source to odd end target data migration speed is usually limited by either cloud provider settings or limits set by the hardware of physical connection. Though, one of the solutions to prevent these restrictions and to expand data migration throughput is to perform data migration from multiple servers concurrently. Even though, BTorrent [6] illustrates that this is rational but processing this data involvement procedure (As It Is) is far from appropriate "for a one-time point-to-point" migration. However, our parallelization data migration combines data segmentation and erasure code, AES, and hash with developing parallel migration capacity and minimizing data transfer delay.

Multiple data transmission develops the processor's total bandwidth; however, it is limited because the "bottleneck" lies in the specific cloud processor's bandwidth. One more comparable system is the parallel transfer where files are prearranged through several input/output processors to minimize storage bottleneck, which allows transmission across several TCP connections concurrently. Another similar system can be applied in cloud storage, but the overheads to rearranged files across numerous storages should to be considered in order to improve the file transmission performance. In our parallelization model, provisional buffer is created on the cloud machine, and these buffers are indicted to as the 'holes'. As the buffer hole is fully occupied, it is then taken away from the list. The entries of theses buffer-holes are then tested with the descriptor-list to confirm if the buffer-hole which has been loaded by the arriving segment is rejected [1].

In [7], multi-hop and multi-path which are two optimization techniques to increase the performance of file transmission across WAN are briefly discovered. In multi-hop path splitting, throughput was improved by exchanging direct connection between start and end point via multi-hopping. On other hand, multi-path technique includes slicing data at the start point and sending it through various overlapping paths. Chunking and rebuilding data theoretically cause data corruption, damage or failure. Thus, our proposed model takes into consideration fault-protection by applying several Reed-Solomon (RS) code values along with the data sizes to supplementary explore the impact of different values.

The authors in [8] propose a cloud-based data management system targeted for big data science applications running across large and high geographical environment sites which shows expectable data transfer management performance in terms of cost and time. The system automatically implements and adjusts performance designs for the cloud platform structure for enhancing data and efficient schedule of the transfer process. Though, it shows a poor execution performance when compared with our proposed parallelization model.

Another similar approach [9] is the parallel data transfer where data are lined across several intermediate nodes that are spawned in order to avoid the bandwidth limitation, but the overheads to reallocated data across multiple nodes in source DC and destination DC have to be taken into attention for performance improvement on the file transfer.

## 3. Server to Cloud Parallel Data Transfer Modelling

### *3.1. General Concept of the Model*

In this section, we define the outline of our server-to-cloud parallel transfer conception and formulate an arithmetic model constructed on the proposed parallel migration concept. In the rest of the paper, we then cover the server-to-cloud parallel migration task as parallel data transfer.

Figure 1 represents the construction overview of our parallel migration model. The process starts with the data as the input, the data is then segmented into chunks, every segmented data is

encrypted. Encrypted chunks are migrated to the cloud and then decrypted. The decrypted chunks are then compared against the un-encrypted segments on the local server. If they don't match, the process is taken back to the segmentation stage. If they match, the segments are then re-assembled and saved on the cloud server. We then end the process.

The following is the arrangement of status for a single file transmission from server to cloud storage.

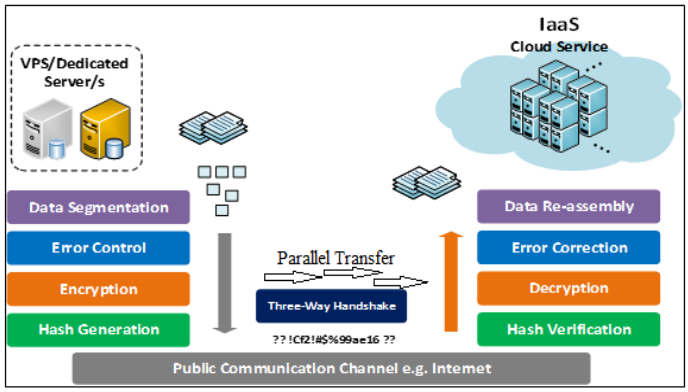


Figure 1: Overview of parallel transfer mechanism

- Data Segmentation [10]: Data segmentations split-up the bigger data units into smaller controllable data slices. In our data segmentation, we will be using an algorithm that will take the shortest time possible to segment larger volumes of data packets, while optimizing the memory efficiency of the working machines. In this algorithm, the system scans the data packets to find the size of the whole data to be segmented. The total data size is then used as a base for the determination of how many segments should be generated from the original data. Data size levels are then predefined to allow the categorization how a given data set belonging to a given size group is segmented. The categorizing of data based on the whole data size ensures that we have larger data size, the right segmentation size will be obtained in order to make the process faster. The data will then be split into smaller equal sized chunks of 4MB each. The chunks are assigned a sequence number each, according to the order in which they are segmented. If we have a larger volume of data, then more instances of segmentations will be initiated to make sure that less time is taken to process the whole data. The number of segmentation instances is dictated by the size of whole data.
- Error control: The end-to-end transmission of data from the server to the destination storage encompasses many steps which can be subjected to errors leading to data loss. Error control ensures that any error that occurs during the transmission of the data is detected and addressed. Reliable data transmission implements error detection and correction. Error correction enables a system to detect and correct both the bit-level and packet-level errors. There are two types of errors that can occur during the data transmission: single-bit error, in which only one-bit changes from 0 to 1 and vice versa and burst error where two or more bits change from 0 to 1 and vice versa. Burst-error is known as the packet-level error which results in duplication or re-ordering of the data. We are using the checksum method for error control which is applied at the upper-layer, and it uses the concept of redundancy. There are two operations which are being performed in checksum technique: Checksum-generator on the server side and Checksum-checker on the cloud side while the local server makes use of the checksum-generator. The following is the procedure for the checksum generation:
  - The data segments are sub-segmented into equal n-bits.
  - This is followed by adding together the sub-segments with the usage of 1's compliment.
  - The data segment is re-complimented to become a checksum.
  - It is then sent alongside the data-unit.
- Encryption: The system uses an Advanced Encryption System (AES) with the RSA key-exchange. The symmetric encryption technique requires a proper key exchange to maintain the security of communication and minimize the risk of eavesdropping during the key exchange phase. RSA-2048 is then used for key exchange. The main reason for not choosing RSA for encryption is that it needs more processing power and is slower than the mighty AES.
- Hash generation: Hash functions are one-way which means that it is impossible to revert from the generated values to the initial values. The hash function is used in our system to enhance the confidentiality of the data (message). We are using the secure hashing algorithm (SHA) which was modeled after MD4 and its proposal was made by NIST to offer a more secure hash standard (SHS). This will produce a 160 bits hash value.
- Three-way handshake initialization: Three-way handshake is a flow control technique which allows the communicating devices to initiate the communication platform. In Figure 2, the local server will initiate the three-way handshake by sending the cloud server a SYN (synchronization) message. This is a way of asking the cloud server if it is ready to communicate. The recipient, which is the cloud server sends back a SYN/ACK message to the local server telling it that it is ready for the communication. The local server then sends an ACK (Acknowledgement), and the connection between the two sides is established. The data is then transmitted.

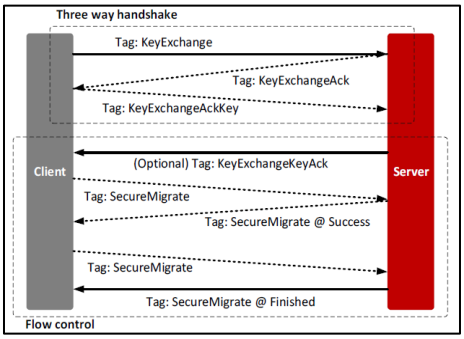


Figure 2: Three-way handshake initialization.

- Hash verification: Hash checks for the integrity of the message. The cloud server receives a segment and runs SHA algorithm over it, if it matches, then we increment the number of received healthy segments by one, we ignore (discard) the rest of the segments which do not match. The system waits until receiving other segments (we know the total number of segments sender

is going to send for us) and see how many of them are healthy (their integrity can be verified).

- Decryption: With the Cipher Block Chaining decryption, application of the inverse-cipher function to the first cipher-text block results into an output-block which passes through the XOR operation with the initialization-vector to allow the recovery of the first plaintext block. The inverse-cipher function is also applicable to the second cipher-text block which provides the output-block which is then XORed with the first cipher-text block for the recovery of the second plaintext block. Generally, the recovery of any plaintext block (except the first plain-text), the inverse cipher-function applies to the corresponding cipher-text block, and the output-block is XORed with the prior cipher-text block.
- Error correction: Error correction allows the cloud server which receives the data to reconstruct the original data in the event that it was corrupted during transmission. Error correction is implemented using the hamming code. Hamming code is a single-bit error correction method that uses the redundant bits. The redundant bits are included on the original data and their arrangement is in such a way that any different and incorrect bits will lead to the production of various error results. The corrupted bits are then identified and upon the identification of the corrupted bits, the cloud server which receives the data is capable of reversing its values hence, correcting the errors. Hamming code applies to any data length and makes use of the relationship between the data and the redundancy-bits. Checksum-checker: The cloud server makes use of the checksum-checker. The following is the procedure for the checksum checker:
    - The cloud server which receives the data-unit will divide the received data-unit into equal sub-segments.
    - All the sub-segments are then added together using the 1's compliment.
    - The outcome is complimented again. If the final result is found to be zero, the system accepts the data while when it is found to be 1, the data should be rejected and considered to have errors.
- Reassembling of data [11]: After all the data segments have arrived and each segment is verified, an algorithm to arrange the segments in order according to their sequence numbers will be used. This ensures that the data segments will be ordered back to their original state before being segmented on the local server [12]. In Figure 3, the flowchart diagram shows how the model receives the segment and allocates the buffer space for each segment. There are more segments in the buffer, by comparing the size of the segment with the size of the space available in the buffer, the segment is taken back. If there are no more data in the buffer, then the data unit is identified, and the data unit is bound to the context. It will then check if it is occupied by a different data unit. The bound data is then updated, and the segment is inserted into the data unit. The number of segments is then checked, if there are any remaining segments, they will return back to step one. If there are no remaining segments, then the header will be updated and the data unit will be transmitted. The numbering sequence plays a crucial role as it is the sequence number used to arrange back the data to its original state [13]. The size of the unencrypted segments on the source server is furthermore tested and matched with the size of the corresponding decrypted segments on the destination server. Re-assembling of data segments will only take place after all the previous stages have been successfully completed.

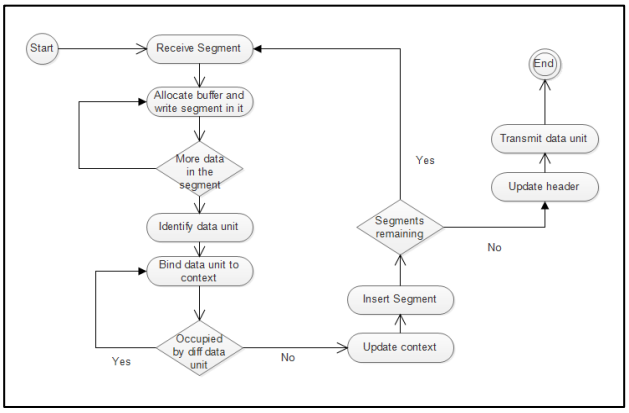


Figure 3:Reassembling of segments.

### 3.2. Parallel Transfer Algorithm

Server user may perhaps definitely benefit from the rapid growth of multicore systems. The physical server typically holds several regular data centers or computer storage, where commodity high-performance multicore bunches are employed. Our migration algorithm designed for one direction data flow which begins at the server storage and ends at the cloud storage. Algorithm 1 shows the parallel-serial transmission which starts from a master processor acting as a management system. The aim of the master processor is to reach the final output by collecting the necessary information from the slave processors. In our migration algorithm, we will assume that the data is ready for transfer after the four stages: data chunks, error control/correction, encryption/ decryption with AES-256 and SHA-256 algorithm which is identified as the significant component of the integrity process. The transfer process will start by taking the ready data list, FList, and divides it between the slave processors. Correspondingly, each slave processor would obtain blocks of FList. We use single program multiple data (SPMD) algorithm where there is a single instruction that each processor performs, but on various data. Each slave processor would establish its secure channel, port, and buffer. Each slave processor will then send its Flist to the cloud buffer before the storage stage. The final task of the slave processor is to update its Flist by removing the sent chunks from the list. This procedure would reflect to have a higher bandwidth transmission capacity. Finally, the master processor receives the FList from all the slave processors and takes the re-transmission decision. If the Flist size=0, it means that all the data chunks have been sent successfully. However, if Flist size=!0, it means that the master processor will request for the re-transmission of the missing chunks by the method explained in [1]. The master processor would close the connection after all the re-transmission requests. At this phase, the cloud buffer is full of the data that need to be checked for integrity, verified and re-build to be stored.

### 3.3. Time Factor of Parallel Transfer

The performance of the proposed model will be evaluated by computing the average of execution time, the number of processors, and the speedup ratio.

**Algorithm 1** Parallel Migration Algorithm

**Input:** FinalChunksList FList

```
procedure AllocatesBlockToEachProcessor
    P ← Number of processors
    if Processor 0 then                    ▷ Processor 0 is the root
        Scan FList size
        Establish Secure Connection (3-way handshake )
        Divide the FList into x by y blocks        ▷ Matrix
        for i ← 0, x − 1 do
            for j ← 0, y − 1 do
                Send block (i, j) to processor (i+j) mod P
            end for
        end for
    end if
end procedure
procedure EachProcessorTransmission
    (Note that RS,AES25,SHA5126 is applied on FList)
    Receive FList blocks from the root
    S ← Scan block size
    for index ← 0, S − 1 do
        Initiate the TCP
        Transfer the block
        Remove the block (FList[index])
    end for
    Returen FList
end procedure
if Processor 0 then
    if FList Non Empty then
        while NotAllFListsSent do
            send the next FList
        end while
    else Close Connections
    end if
end if
```

### 3.3.1 Data Distribution Time (Segmentation+ CRS+ AES + HashGen)

Divided into four actions, file chunking, encoding, encryption, and hash generation. In order to minimize the delay between individual execution time, transferring has to be started directly after the segmentation when the exact chunk is ready, without waiting for the overall segments process to be completed. Server execution time can also be minimized by manipulating the chunk size and sequence when the network resource is unutilized.

### 3.3.2 Parallel Data Migration Time

This time factor indicates the transfer from local server to buffers in the cloud storage. In order to increase the throughput of this phase, the parallelism method can be applied only if the transmission files are of a large number. Processor 0 would act as the root and starts to define the number of the chunks and divide them among the slave processors. As a result, each slave processor would collect a block of chunks. Later, they would establish their connections and send their chunk lists to the cloud buffers. The control would then be passed to TCP in order to finalize the transmission task.

### 3.3.3 Data Amalgamation Time (HashVerify +AES +CRS +Re-Assembling)

In this time factor, the file reassembling is the prime task. It also involves the migration from the cloud buffers to the destination storages. This step also contains massive disk operations such as the writing process.

### 3.4. Model and Evaluation of Parallel Migration

The parallel performance was analyzed by displaying speedup rate and throughput to illustrate the parallel model profit. The speedup of parallel operations on P processors is defined as the following [14]:

$$S_P = \frac{T_1}{T_P} \tag{1}$$

Where $T_1$ is the sequential execution time using one processor, $T_P$ is the parallel execution time using P processors, p and 1 are indexes as written in equation 1. While Sequential Transfer Time equation ($T_1$) is:

$$T_1 = \frac{D_s}{t_e} \tag{2}$$

Where $D_s$ is data size in MB, $t_e$ is transfer speed in MB/s. Parallel Transfer Time ( $T_P$ ) = data distribution time in the server + transfer time + data amalgamation time in the cloud. Hence, the speedup rate is acquired as below:

$$S_P = \frac{D_s/t_e}{\frac{T_P}{D_s/t_e + D_s(1/P)/t_e}}$$

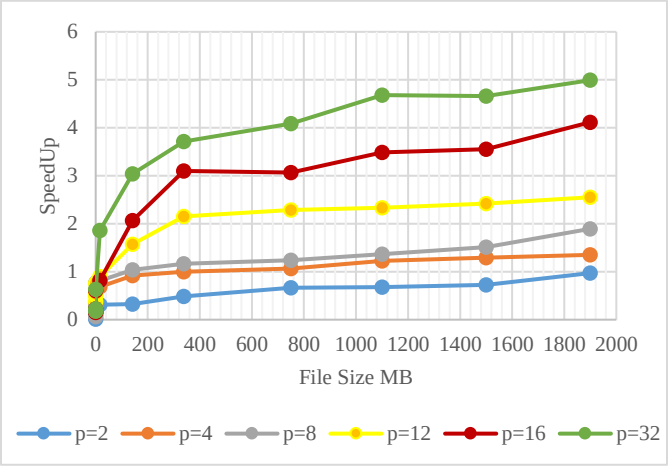


Figure 4: Speedup model for parallel transfer, where p is the number of processors.

It can be noticed that the smaller file transfer size (not many of small and large files together), a minimum number of parallel processors are required to accomplish a speedup. However, when trying to parallelize the transfer, there is no prediction of when the packet loss rate would begin to increase exponentially which causes a critical overhead. Thus, smaller file sizes do not benefit from the parallelization. Therefore, Figure 4 shows that the speedup rate is steep for the file size of below 4MB.

For a larger data transfer size, the speedup rate grows as the number of parallel processors are increased. However, performance gains are hardly worth when each subsequent increase in processors number brings a lesser performance profit contrasted to the previous increment.

## 4. Experimental Evaluation

This section shows the performance evaluation for our parallel migration system on an Infrastructure as a Service (IaaS) cloud environment.

### 4.1. Experiment Setup

We carry out these experiments on two separate VMs (VM hardware were identical) hosted on Cray XC40 within two locations: Oregon and Amsterdam. Linux Operating System in various cores with 8 GB of memory was assembled on the VMs. All the encryption /decryption stages are constructed with the help of python cryptography 2.1.4.

Our experiment concentrates on migrating a single file from the server to cloud storage. The model is designed in such a way that each part of the system is pluggable. Thus, replacing an encryption algorithm will not affect the selection of the coding algorithm. We have chosen numerous file sizes, starting from .05 MB to 1900MB and RS parameters (k, m) in range of ($1 \leq m \leq k$ and $k \leq 256$, where m=parity) in order to evaluate the transfer rate. For coding layer, we employed PyEClib source on GitHub (1.5.0) module with liberasurecode source on GitHub (1.5.0) as Interface to erasure back-ends for Reed-Solomon coding.

### 4.2. Preliminary test

To evaluate the actual performance, we first examined the performance of our algorithm by measuring the execution time in the server (source) and cloud storage (destination) processes for a data, depending on the file size and variability of RS parameters k and m.

In Table 1 below, we showed the average operating time for the source and destination for each process. It is quite apparent that the source average is mainly estimated by the slicing process because it involves additional parameters and computations than the other internal operations.

Table 1: Average execution time in the source/destination.

| OPERATIONS | TIME (ms) | SUM |
|---|---|---|
| CRS Coding | 691.99 | 1,089.74 |
| AES Encryption | 116.78 | |
| SHA Hash | 280.97 | |
| SHA Verify | 277.39 | 487.55 |
| AES Decryption | 119.53 | |
| CRS Decoding | 90.63 | |

The throughput define as that is the amount sum of processed data that execute in a period of time (processed data /second) [15]. It is concluded as the following relation:

$$P_n = \frac{W}{T_P} \qquad (3)$$

Where W is indicated as the workload, $T_P$ as the execution time required to accomplish the calculation using P processors [16], p is index as written in equation (3). From equation 3, the throughput for each location is shown in the Table 2. Table 2 depicts the comparison between the average throughputs of transmitting many file sizes between Oregon and Amsterdam locations. The transfer throughput gained for Oregon and Amsterdam is ~.9, .7 MB/ms respectively. The range of throughput achieved when transferring the file is astonishingly large. It was observed that the throughput rate does not follow a certain way; the variation seems to arise randomly.

Table 2: Locations Transfer Throughput.

| File sizes | Oregon TO Amsterdam | Amsterdam TO Oregon |
|---|---|---|
| **0.05** | 0.005101482 | 0.001007 |
| **0.10** | 0.009208418 | 0.001696 |
| **0.55** | 0.038767525 | 0.008251 |
| **1.00** | 0.019112421 | 0.011328 |
| **17.00** | 0.041170096 | 0.073851 |
| **142.00** | 0.086551034 | 0.263185 |
| **339.00** | 0.093888415 | 0.333905 |
| **750.00** | 0.107918311 | 0.442913 |
| **1,100.00** | 0.102385909 | 0.468758 |
| **1,500.00** | 0.109188015 | 0.502546 |
| **1,900.00** | 0.114588619 | 0.579061 |

### 4.3. Result

Figure 5 and Figure 6 are linearly dependent features and are just shown for further clarification. It is verifiable that by choosing bigger fault tolerance rate in the coding stage (smaller k in proportion to m) the final transfer time increases rapidly. It is shown that transferring bigger files with no-error tolerance (k =m) added in the coding stage, the system will exhibit the best performance, especially in case of bigger file sizes. Choosing k such that it covers 10-15% error correction without the need for data retransmission in smaller file sizes less than 24MB, the impact on transmission time is noticeably less than the profit it brings.

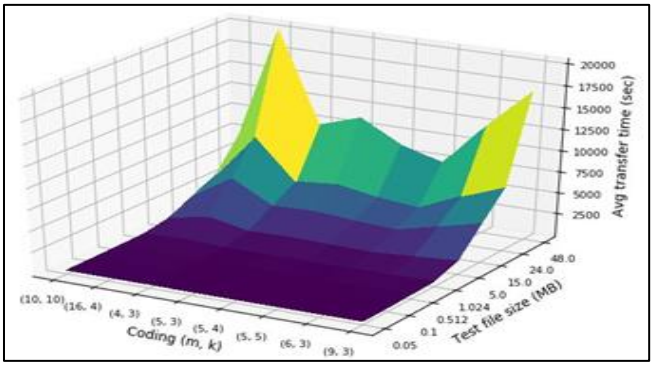


Figure 5: Average execution time in source side.

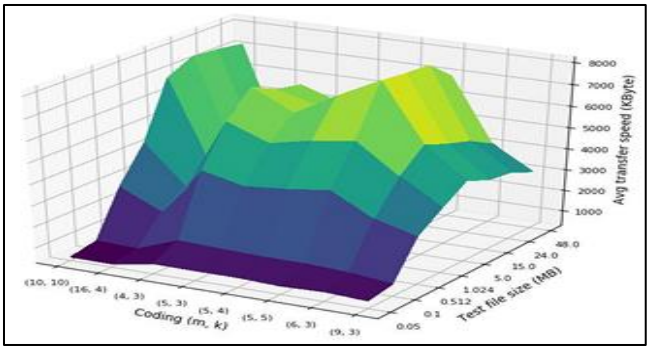


Figure 6: Average execution time in destination.

In Figure 7, we portray a 2D graph of the speedup rate against data size for a different number of processors. It can be noted that analogous to our model, speedup cannot be obtained for the data migration that is below 0.5GB by using 2, 4, 8, and 16 parallel processors. The overall note is steady with our approach, the speedup is extremely poor for small file size, but increases for bigger file size.

Although the speedup rate is not computed yet in the real environment, we found an independent argument that can be used in our model to compute the speedup in the real environment. Based on our observation, the independent argument is close to 0.47. Hence, we are certain about the fact that when the count of parallel processors is greater than 8 and the data size is bigger than 1GB, the speedup will be achieved.

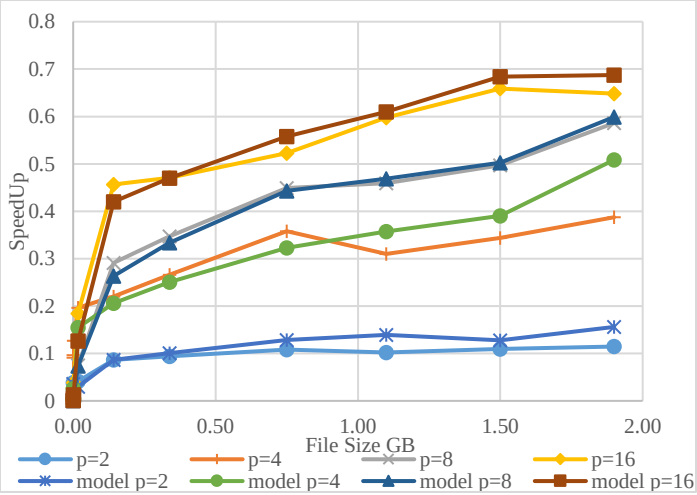


Figure 7: Comparing actual speedup and our model for parallel transfer, where p is the number of processors.

## 5. Conclusion and Future Work

This paper concentrated on explaining and exhibiting a parallel design for data transmission in cloud environment. First, we demonstrated the proposed model. Next, we presented the implementation and evaluation of the proposed model. Though our parallel transfer model shows a prospect to enhance the performance, however, our implementation exposed that there is an independent argument that should be considered. This study also suggests that there are overhead factors that can affect the parallel transfer performance, such as initialization, chunking the data file, and transfer time. Thus, those overhead effects should be investigated in order to reach maximum performance. In brief, we should consider the computation of hardware power and processors I/O's speed that is assigned by the CSP to design the parallel migration model.

## Conflict of Interest

The authors declare no conflict of interest.

## References

[1] M. Alkhonaini and H. El-Sayed, "Optimizing Performance in Migrating Data between Non-cloud Infrastructure and Cloud Using Parallel Computing," *2018 IEEE 20th International Conference on High Performance Computing and Communications (HPCC)*, Exeter, United Kingdom, 2018, pp. 725-732.

[2] Hesham El-Rewini, Hesham H. Ali, and Ted Lewis. 1995. Task Scheduling in Multiprocessing Systems. *Computer* 28, 12 (December 1995), 27-37. DOI: https://doi.org/10.1109/2.476197

[3] S. Razdan, Fundamentals of Parallel Computing. New Delhi: Alpha Science International, 2014.

[4] "rsync(1) - Linux man page." [Online]. http://linux.die.net/man/1/rsync. 2015.

[5] H. Pucha, M. Kaminsky, D. G. Andersen, and M. A. Kozuch, "Adaptive File Transfers for Diverse Environments," in USENIX 2008 Annual Technical Conference on Annual Technical Conference, Berkeley, CA, USA, 2008, pp. 157–170.

[6] Bram Cohen, "The BitTorrent Protocol Specification." [Online]. http://www.bittorrent.org/beps/bep_0003.html. 2015.

[7] G. Khanna et al., "Multi-hop Path Splitting and Multi-pathing Optimizations for Data Transfers over Shared Wide-area Networks Using gridFTP," in Proceedings of the 17th International Symposium on High Performance Distributed Computing, New York, NY, USA, 2008, pp. 225–226.

[8] R. Tudoran, A. Costan, R. Wang, L. Bouge, and G. Antoniu, "Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers," in Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, 2014, pp. 92–101.

[9] C. B. M. Lek, O. B. Yaik and L. S. Yue, "Cloud-to-cloud parallel data transfer via spawning intermediate nodes," TENCON 2017 - 2017 IEEE Region 10 Conference, Penang, 2017, pp. 657-661. doi: 10.1109/TENCON.2017.8227943

[10] M. Alkhonaini and H. El-Sayed, "Minimizing Delay Recovery in Migrating Data between Physical Server and Cloud Computing Using Reed-Solomon Code," *2018 IEEE 20th International Conference on High Performance Computing and Communications (HPCC)*, Exeter, United Kingdom, 2018, pp. 718-724.

[11] M. Alkhonaini and H. El-Sayed, "Migrating Data Between Physical Server and Cloud: Improving Accuracy and Data Integrity," *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, New York, NY, 2018, pp. 1570-1574.
doi: 10.1109/TrustCom/BigDataSE.2018.00226.

[12] "Sequence data submission and accession numbers", FEBS Letters, vol. 360, no. 3, pp. 322-322, 1995.

[13] C. Diot and F. Gagnon, "Impact of out-of-sequence processing on the performance of data transmission", Computer Networks, vol. 31, no. 5, pp. 475-492, 1999.

[14] Roosta, S. " Parallel Processing and Parallel Algorithms: Theory and Computation ", ISBN: 0-387-98716-9, Springer Verlag, 2000.

[15] Kermarrec, A.; Bougé, L.and Priol, T. "Euro-Par 2007 Parallel Processing ", 13th International Euro-Par Conference: Lecture Notes in Computer Science, ISBN 978-3-540-74465-8, Vol. 4641, 2007.

[16] Hwang, K. and Xu, Z. (1998). Scalable Parallel computing. New York: McGraw-Hill.