ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(45), DOI: 10.17485/ijst/2016/v9i45/95582, December 2016* ISSN (Online) : 0974-5645

Effective Data Transfers through Parallelism in Cloud Networks

Sruthi Anand* and Sornalakshmi Krishnan

Department of Information Technology, SRM University, Chennai - 603203, Tamil Nadu, India; Sruthi.sruth31@gmail.com, sornalakshmi.k@ktr.srmuniv.ac.in

Abstract

Objectives: We aim at analyzing a method that enhances throughput for huge heterogeneous file transfers in the inter cloud and intra cloud for data transfers. **Method:** The proposed work identifies the files to be transferred in the cloud, splits the data packet into chunks and pushes them to the cache storage from where they are transferred onto the destination cloud. This method helps in enhancing the throughput of the data being transferred and simulations are observed. **Findings:** Generally, the previous methods focused on considering the file for being large or small and then predicting to use pipeline or parallelism. In this work, we transfer the file irrespective of the size by splitting it into a reasonable chunk of data for effective utitilization of the available bandwidth. **Application/Improvements:** Consideration with large and small files and then splitting them takes more time with chances of data being lost or not utilized. Hence, our work features more on assuring that the data is being sent to the cloud with no data loss.

Keywords: Big Data Transfer, Heterogeneous File, Inter-Cloud Transfer, Parallelism, Throughput

1. Introduction

File transfer plays a very important role in the cloud environment as the transfer is expected to have a good quality. Moreover, efficient and flexible file transfer with reliability has an important role in guaranteeing a good quality of service for users. In the recent times, the data produced by large scale applications are higher and hence needed to be stored in cloud that are reliable and efficient. In such a case, when data is being transferred, it is necessary to know that throughput increases. When data transfer takes place between systems, it is in practice to check for the system level requirements. In networks, although there are protocols being developed, a protocol becomes inefficient because of the end system characteristics thereby resulting in underutilization of the protocol. Hence, we make a joint consideration of the parameters, such as NIC capacity, memory, background traffic etc. and to perform it in the cloud environment we also make use of two techniques, parallelism and pipelining[1]. Most of the cloud applications are designed to move the data files either between the cloud storage or from a system to a cloud.

The importance of optimizing data transfers between cloud is increasing with an exponential growth of data traffic. Lossless data compression can be essential in increasing communication throughput, reducing communication latency, achieving energy-efficient communication and making effective use of available storage. During the transfer, it is necessary to make considerations of heterogeneous files. In cases before, the transfer of heterogeneous files were done based on two parameters: A larger file and a smaller file[2].

Taking into account these two conditions, based on file size, the previous work focused, implements the transfer of larger files using the technique of parallelism and the smaller files by pipelining[3]. But to analyze a file to be larger or smaller and decide the technique to be implemented may not be reliable and efficient. Also the order in which the packets are sent can also be misplaced[4].

* *Author for correspondence*

This takes place by streamlining the number of packets by optimizing the optimal number of parallel streams. Some algorithms were implemented using GridFTP transfers[5].

Generally, when the transfer is done, there can be heterogeneous files, meaning large and small files. So when this is done, it may cause underutilization of the bandwidth for small files, thereby resulting in data loss. Hence, these transfer mechanisms were implemented by fixing a statistical threshold for parallelism and pipelining, which was time-consuming and difficult. In the proposed work, we combine these two techniques and based on the replacement algorithmic value we transfer the data. These can be done to improve the throughput using multiple data paths between the systems. Pipelining is used to send large number of small files, thereby resulting in no idle channel[6]. Parallelism results in sending multiple portion of the same file to have high throughput giving an unfair share of the bandwidth. Thus, a comparison with normal file and the proposed transfers we transfer techniques we achieve a high throughput. With the transfer of data taking place between networks, it is vital to consider the transfer that is happening in cloud networks i.e. inter cloud or intra cloud. By trends, cloud computing plays a vital role in transfer and storage.

2. Motivation

Cloud computing has a significant role to play in data storage and retrieval with the amount of data generated and networked. Taking the concept of parallelism which was introduced in networks onto cloud and determining the throughput when large heterogeneous file transfer happens in cloud is interesting. The main goal of using parallel streams of data is to make the best use of the resources and bandwidth and have an increased throughput[10]. Combining pipelining and parallelism, pipelining caters to the need of transferring small files with multiple transfer commands queuing up at the server and reducing the delay between transfer completion and its receipt[11]. In this paper, we clearly show how to best utilize the concept of parallelism and pipelining to optimize the transfer of a large data set on an intra cloud architecture. These parameters of parallelism and pipelining help us to determine the ultimate throughput and network utilization obtained by many data transfer applications. So when considering the transfer of data in intra cloud, automated transfer requires the use of parallelism to acquire optimization of throughput in the cloud environment. But a traditional file transfer mechanism implements with considerations of its quality of service, protocols, number of CPU chores, etc[12]. Hence, the existing work may not be able to utilize the bandwidth available efficiently, which has been overcome in the proposed.

The practical way of increasing the throughput is to have multiple streams of data. The fact that the use of multiple parallel streams may produce better performance than using a single stream. The transfer should be based on a single instant feedback. In this paper, we achieve optimization by having optimal number of parallel streams to enhance throughput. By doing this, we take into account that all the parallel streams would be uncongested without data loss. This is where parallelism becomes effective. When data transfer takes place between the cloud environment, considerations of loading large data files and bigger size data, best suit parallelism. Generally, parallelism happens in TCP sockets where the network becomes congested as they arrive in an out of order split and increase packet loss. In order to implement such methodologies in big data transfers, say heterogeneous files, taking the concept of parallelism against pipelining will enhance the throughput of data packets that are being automatically transferred between the cloud networks. Pipelining was done for the small files in the previous work, which in our case would not explicitly require a special parameter to classify the small files. The findings from various papers were:

The previous techniques used in cloud networks with the use of parallelism were to divide the data, consisting of large and small files. But to perform the data transfer by setting up parallelism level for each transfer is difficult[1]. In the case of GridFTP transfers, technique of pipelining was implemented and this took care of only about the order of data. But with many parallel streams, chances for the data to arrive out of order was of high possibility[3]. When it was implemented on WAN networks, parallelism was done, to choose a parallelism level and decrease data size for error correction[5].

3. Proposed Architecture

To implement this data transfer using parallel streams, we

consider two ideas: 1. To split the large file into a smaller number of packets or chunks and then 2. Accommodate it in a buffer. This buffer manages all the incoming parallel streams of data. The process of split takes place when larger files that are greater than 1000 MB are to be transferred. And this is where split takes place to parallel streams of data.

Algorithm 1: Split and Merge

Require: *list_0f_files ,tot_no_of_files, source path*
Determine: *the file and its source path*
Calculate: *the mean _file_size*
Read: *themin_chunk_size*
If: *min_chunk_size<sourcefile*
Then
Do Split(file)
Split = file/min_chunk_size
Split <--- No. of_chunks
Generate chunk(ID) , file ID
Do
Push chunks to cache
Cache ← chunk + chunk(ID)+ file_ID
Merge(chunks)
Merge = in order _ arrival
If *out_of_order then ascend to chunk_ID*
end.

Next, it is necessary to know the size of the cache before placing them in the cache ,which is determined using the formula.

Size = log2 (B*Smemory/Scache*Ssize).

Where,

B = the bandwidth of transfer.

Smemory = range of operating memory (RAM).

Scache = size of the cache indicating the word count .

Ssize = Server size.

In computing, cache algorithms are those that a computer program or a hardware maintained structure can follow to manage a cache of information stored on the computer. When the cache is full, this algorithm chooses which items to discard making room for the new ones.

Algorithm 2: Cache Placement Algorithm

Require: *Cache_size, available_space, size _of_file, no_of_chunks*
Read: *no._of_chunks*
Read: *available_space*
If: *no._of_chunks<available_space*
Do
Cache ← push chunks until full
Else
If *no._of_chunks> available _space*
Then
Fill cache with available chunks
Then
Wait ← until cache free
Time_set ← cache_refresh
Do
Push chunks to cache
End

Cache Refresh:

The average memory reference time is given by:

$$T = m * T_m + T_h + E$$

Where,

T = average memory reference time

M = miss ratio1-(hit ratio)

T_m = time to make a main memory access when there is a miss

T_h = the latency, the time to reference when there is a hit

E = various secondary effects, such as queuing effects in a multiprocessor systems.

There are two primary figures of merit of a cache: The latency and the hit rate. There are also a number of secondary factors affecting cache performance. The "hit ratio" of a cache describes how often a searched-for-item is actually found in the cache. The efficient replacement policies keep track of more usage information in order to improve the hit rate.

The "latency" of a cache describes how long after requesting a desired item the cache can return the item (when there is a hit). Faster replacement strategies typically keep track of less usage information or in the case of direct-mapped cache, no information to reduce the amount of time required to update that information. Figure 1 shows how data transfer is being carried out in the cloud networks. Figure 2 depicts the flow of our proposed system.

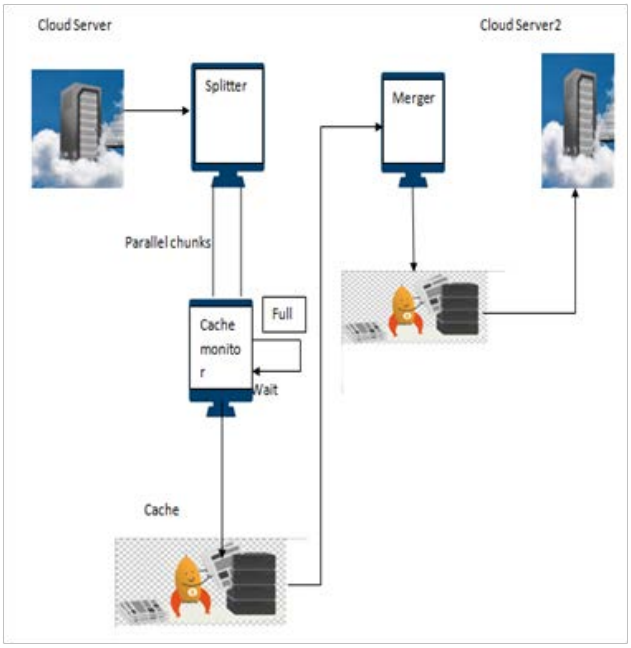

Figure 1. System architecture.

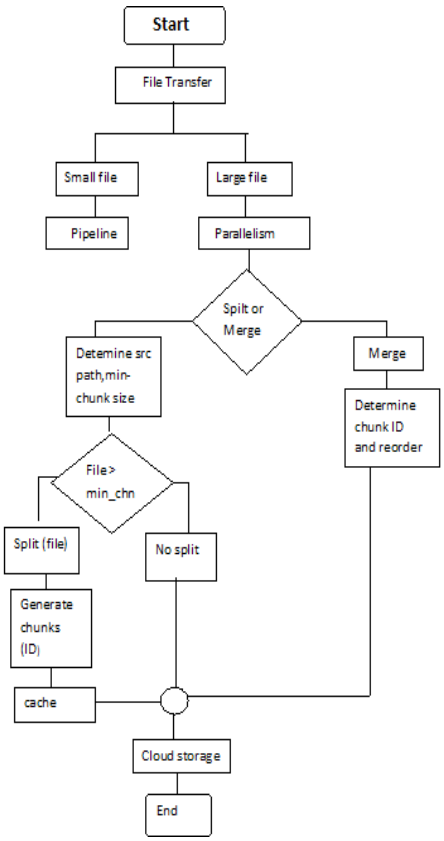

Figure 2. Flowchart.

4. Experimental Result

Thus, this implementation shows that the data transferred between the cloud provides higher throughput. They are implemented through the use of cloud sim to show the simulation of the data packets that are being transferred. Figure 3 shows the time taken to transfer a data using existing methodology. The time taken by a packet to reach the cache and then to the destination server from the cache is recorded by means of a graph. The scenario is being considered with various heterogeneous files and the comparison is made between the normal file transfer and the algorithm implemented transfer to show the best results. The graph here shows the time taken by each cloudlet from the virtual machine to reach the destination, as shown in Figure 4.

By implementing these algorithms onto the cloud, we infer that the time taken for the transfer in cloud networks is much lesser than the already existing transfer mechanisms, shown in Figure 5.

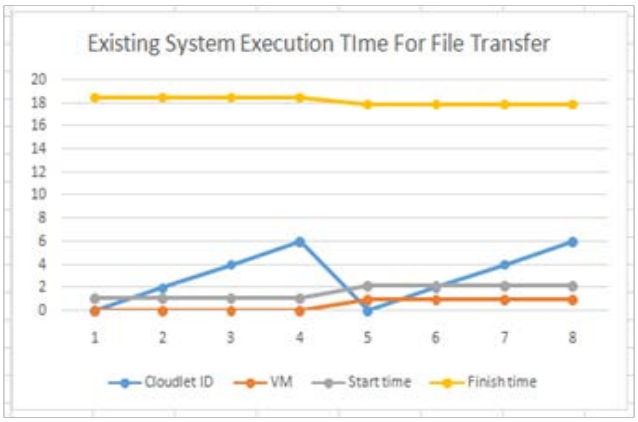

Figure 3. Existing system file transfer.

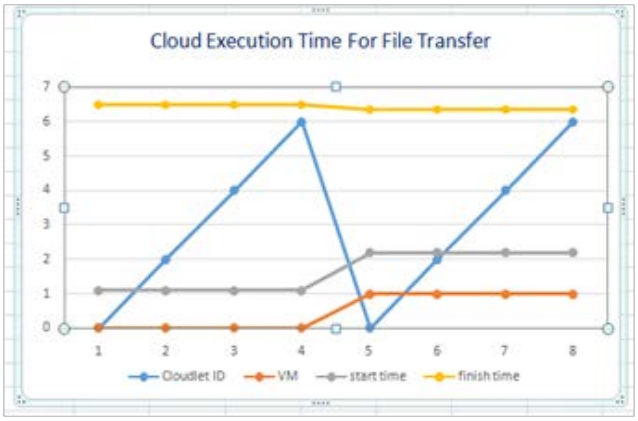

Figure 4. File transfer in cloud.

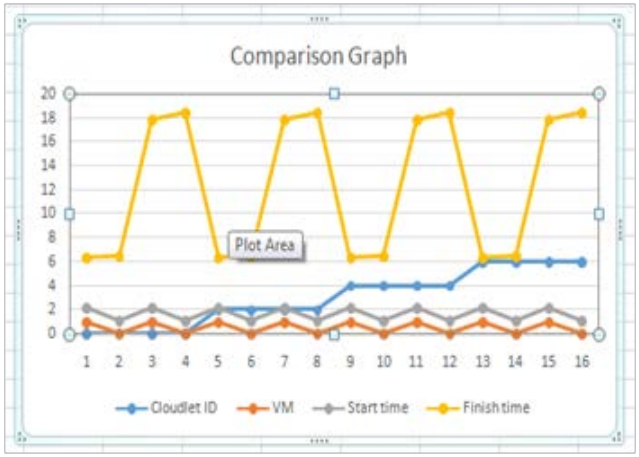

Figure 5. Comparison graph.

5. Conclusion and Future Work

The data transfer between the cloud is implemented using the techniques of parallelism and pipelining by the help of a buffer, which helps us to accommodate the data packets as and when they are received, so that they need not wait for the acknowledgment to be received. By this method, an increased throughput is achieved during the transfer compared to the normal file transfer. This work also focuses on ensuring that all the data packets that are sent make effective utilization of the protocol and bandwidth, by splitting the data file to small data chunks. Hence, our algorithms enable to identify the data packet according to its size and then make the transfer. To make the effective utilization of the bandwidth and thereby not resulting in any protocol inefficiency for the packets, the concept of parallelism and pipelining best suits. These algorithms can also be implemented as an optimization service. In the future, we could also consider factors of concurrency make it more efficient and include heterogeneous files of media such as audio files, video files, etc.

6. References

1. Fang X, Veeraraghavan M. A hybrid network architecture for file transfers. IEEE Transactions on Parallel and Distributed Systems. 2009; 20(12):1714–25.
2. Hacker TJ, Noble BD, Atley BD. The end-to-end performance effects of parallel tcp sockets on a lossy wide area network. Proc IEEE International Symposium on Parallel and Distributed Processing (IPDPS'02); 2002. p. 434–43.
3. Yildirim E, Arslan E, Kim J, Kosar J. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. IEEE Transactions on Cloud Computing. 2016 Jan-Mar; 4(1):63–75.
4. Altman E, Barman D, Tuffin B, Vojnovic M. Parallel tcp sockets: Simple model, throughput and validation. Proc IEEE Conference on Computer Communications (INFOCOM'06); 2006 Apr. p. 1–12.
5. Arslan E, Ross B, Kosar R. Dynamic protocol tuning algorithms for high performance data transfers. Proceedings of the 19th International Conference on Parallel Processing ser Euro-Par'13; 2013. p. 725–36.
6. Lining Z, Yunlan W, Jianhua GU, Tianhai Z. Adaptive file transfer and policy study in cloud computing. 2011 IEEE International Conference on Intelligent Computing and Integrated Systems (ICISS); 2013 Jan 1-8.
7. Yildrim E, Yin D, Kosar T. Prediction of optimal parallelism level in wide area data transfers. IEEE Transactions on Parallel and Distributed Systems. 2011; 22(12):1–14.
8. Yildirim E, Kosar T. End-to-end data-flow parallelism for throughput optimization in high-speed networks. Journal of Grid Computing. 2012; 10(3):395–418.
9. Zaghloul SS. The mutual effect of virtualization and parallelism in a cloud environment. Conference AFRICON; 2013 Sep 9-12.
10. Kim J, Yildirim E, Kosar T. A highly-accurate and low-overhead prediction model for transfer throughput optimization. Proceedings of ACM SC'12 DISCS Workshop; 2012.
11. Choi KM, Huh E, Choo H. Efficient resource management scheme of tcp buffer tuned parallel stream to optimize system performance. Proc Embedded and Ubiquitous Computing; Nagasaki, Japan. 2005 Dec.
12. AnuKarpaga S, Muralidharan D. High throughput pipelining NoC using clumsy flow control. Indian Journal of Science and Technology. 2016 Aug; 9(29). DOI: 10.17485/ijst/2016/v9i29/91236.
13. Varthini S, Muthaiah R. Digital infinite impulse response filter with floating point multiply accumulate circuit using pipelining. Indian Journal of Science and Technology. 2016 Aug; 9(29). DOI: 10.17485/ijst/2016/v9i29/90907.
14. Thiriveni GV, M. Ramakrishnan M. Distributed clustering based energy efficient routing algorithm for heterogeneous wireless sensor networks. Indian Journal of Science and Technology. 2016 Jan; 9(3). DOI: 10.17485/ijst/2016/v9i3/80493.
15. Sasikumar R, Ananthanarayanan V, Rajeswari A. An intelligent pico cell range expansion technique for heterogeneous wireless networks. Indian Journal of Science and Technology. 2016 Mar; 9(9). DOI: 10.17485/ijst/2016/v9i9/67610.
16. Bagheri R, Jahanshahi M. Scheduling workflow applications on the heterogeneous cloud resources. Indian Journal of Science and Technology. 2015 Jun; 8(12). DOI: 10.17485/ijst/2015/v8i12/57984.