RSYNC Algorithm Limitations and Potential Solutions

Sharma Vaishali, Bezzi Marco, Walsh Kaitlyn

I. Problem Background

RSYNC is prevalent for efficient file synchronization in real-time as it enables administrators to propagate changes to files or directory trees. It provides a faster and more effective way for synchronizing the files copies and managing distributed replicas (Rasch and Burns, 2003). It is an efficient algorithm for downward synchronization of files and is used to synchronously update files and directories of source and target computers, and to properly utilize different blocks in the lookup file to reduce data transmission (Develop Paper, 2019). The main feature of rsync algorithm is incremental transmission, which means it will only transmit the changed parts. It moves the minimum amount of data from source to target and saves bandwidth and time by large amount (Rasch and Burns, 2003).

The algorithm is popular for updating files and copying changes over remote servers, there is always room for improvement. Previous studies have looked at ways of improving the efficiency of this algorithm using different alterations. Irmak et al. (2005) evaluate various methods to improve the synchronization of files, based on rsync. The three authors believe that basic performance of the algorithm can be improved based on block size, number of blocks the size of file modifications (Irmak et al., 2005). While they discovered that there is no optimal block size, using variable block sizes via the Karp-Rabin fingerprint method, the synchronization can occur without comparing hashes of the new file to all of the hashes in the old one, but rather you can compare the hash to the corresponding blocks for each file, increasing efficiency (Irmak et al., 2005). This article also talks about the use of erasure codes, to minimize the multi-round protocol of rsync into a single round protocol. At Johns Hopkins University (Rasch and Burns, 2003), discuss a method of improving the rsync algorithm by having the reconstruction of files occur in place, creating ip-rsync. This method releases the temporary space normally used by the rsync algorithm and performs the changes to the file in the space currently occupied. Their method is good for systems with limited space. Numerous methods and algorithms were proposed by various researchers in order to maximize the file synchronization efficiency and data transfer security and privacy. In the present work, three potential limitations of RSYNC algorithm were identified and a few approaches or alternatives were proposed to achieve the desired benefits such as handling of complex synchronization process, computation cost, space management, block based architecture etc.

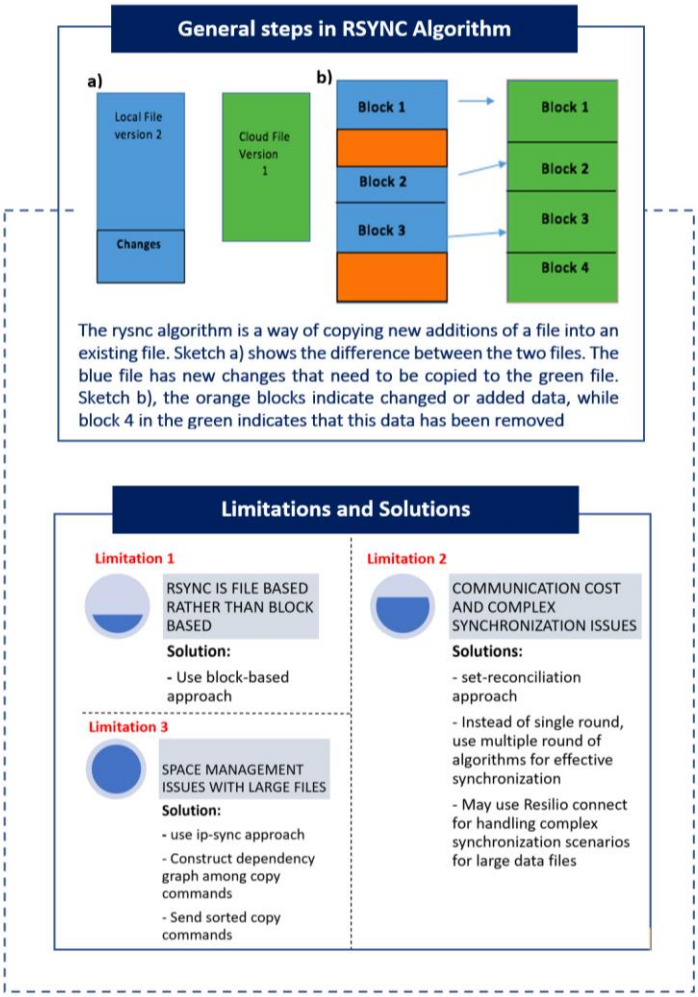

Figure 1. RSYNC Conceptual diagram, some limitation and proposed solutions by the team

Problem Statement and Research Questions:

The problem is to identify the limitations in the existing RSYNC algorithm (shown in Figure 1) and find potenatial solutions or alternative to improve them. To perform this task, several research questions were identified and noted below:

1. How to address the absence of alignments between matching blocks in two files?
2. How to reduce the communication cose and deal with complex synchronization processes?
3. What are the issues with large files and how to perform space management?

II. PAST WORK

Remote file synchronization (rsync) has been studied extensively over the past few years and many methods or approaches were proposed by the researchers/scientists. Many authors addressed the shortcomings of existing RSYNC algorithm and produced modified and novel algorithms for effective and timely synchronization of datafiles. One specific shortcoming as proposed by Rasch and Burns (2003) is that RSYNC cannot back up or replicate block devices so it cannot be used without sufficient temporary space for two copies of a file. A novel in-place rsync or ip-rsync was then introduced by Rasch and Burns (2003) which eliminates the need for additional storage by using the space already occupied by the file. So, instead of using temporary space, the changes to the target file takes place in the space already occupied by the current version. It was helpful in preventing the copy of corrupted data and minimizing the compression loss. Yan et al. (2008) discussed existing two approaches i.e. single-round and multi-round protocols of file synchronization across machines. Single-round is more suitable for scenarios which involve small files and large network latencies, however in case of large files or collections and slow network it may be more preferable to use multiple rounds to reduce communication costs and number of protocols (Yan et al., 2008). A novel single-round algorithm based on the use of *"set reconciliation technique"* is proposed by them for file synchronization that achieves savings in bandwidth consumption while preserving many of the advantages of the rsync approach. With the advent of pervasive IoT, large amount of data is generated and transmitted, so in such scenarios new challenges are opening for network management and information exchange rule (Petroni et al., 2018). A method is proposed by Wang et al., 2018 , which discusses the architecture for data synchronization based on fog computing, RSYNC. A performance prediction model was built to study the effect of cluster configuration parameters on the performance of IoT operations. Sun et al. 2020, studied the problem of local optimization of data during the execution of data scheduling in IoT clusters and presented a novel-data scheduling optimization algorithm to handle the issues of load imbalance and data matching deviation during data scheduling process of IoT. Petroni et al., 2018 addressed the problem of increasing *data volume*, leading to data overloading and data traffic consumption increase when serving a huge number of devices. Edge computing is the approach that the authors discussed in order to move large part of computing close to the data sources and offering benefits such as latency reduction, bandwidth optimization and security. They proposed a decentralized IoT cloud framework where devices connect to the data center through an IoT gateway, also discussed a mechanism for effective data synchronization that uses *Octodiff* (popular data compression tool). This architecture was effective in reducing the large traffic amount generated by IoT cloud-services. So time to time, many algorithms or solutions were built on existing rsync approaches for effective data synchronization. In the present work, the authors have proposed innovative solutions to the limitations of existing rsync algorithms and identified better solutions for remote file synchronization.

III. POTENTIAL SOLUTIONS

SOLUTION STUDY-I

RSYNC IS FILE BASED RATHER THAN BLOCK BASED

The goal of rsync algorithm is to split files into non-overlapping fixed-sized blocks and use hash functions to compute the hashes of the blocks (Tridgell, 1996). The hashes are forwarded to the recipient machine which attempts to find the corresponding blocks in its own file (Suel et al., 2004). One limitation observed in the *rsync* algorithm is the absence of alignment between matching blocks in the two files. It can be solved by matching received hashes with every substring of the same size, rather than the corresponding block in the other file. To increase efficiency, the hashes are composed of two distinct hash functions; a fast but unreliable, and a reliable but more expensive to compute (Suel et al., 2004).

RSYNC functions at the level of files and directories. It examines every file and creates an index of the path to files and directory on the server. This can be a burden for servers with voluminous files since it consumes a large amount of resources in the memory. It also consumes more time to navigate through the directory tree to list every file eligible for backup (R1Soft, 2012). This can be solved by Block Based backup softwares such as Continuous Data Protection (Table 1) which evade the file system to read low level volume blocks.

Table 1. Differences between RSYNC and Continuous Data Protection (Block Based backup Software)

SN	Parameters	RSYNC	CDP (R1Soft)
1	File or Block level	File level	Block level
2	Backup Length	Hours / Days	Minutes when CDP is in sync
3	Backup Method Used	Incremental (One copy is left)	Virtual Full
4	Real Time Backup	No	Yes
5	Reduce Disk I/O resource consumption	No	Yes

The benefit of evading the file system is that it does not affect the backup performance due to the large volume of files. In addition, Block based backup softwares read blocks in the order in which they are on the disk, rather than how they appear in files which are usually heavily fragmented, causing the disks to consume more time seeking data (R1Soft, 2012). Continuous Data Protection also solves the RSYNC time and

Disk I/O resource consumption needed to process deltas using checksums. RSYNC operations usually last longer when a large volume of data is present, no matter how little data has changed. CDP solves this problem using a proprietary device driver able to track changes to the disk at an extremely low level while the system is running (R1Soft, 2012).

SOLUTION STUDY-II
A SOLTION FOR HIGH COMMUNICATION COST AND COMPLEX SYNCHRONIZATION SCENARIOS

Yan et al. 2008 studied single-round algorithms based on set reconciliation for remote file synchronization, that resulted in significant bandwidth saving over previous methodologies or solutions. The technical preliminaries as discussed by authors are: **The *rsync* Algorithm**: The algorithm splits file into blocks and use hash functions to compute fingerprints of the blocks (of size b), these hashes are then sent to the other machine or servers (which looks for similar blocks in its own file). The complete algorithm is shown in the Figure 2. There are two sides: client side and server side. At first block partitioning will be done, and for each block a hash value will be computed and communicated to the Server side. At the server side, for each retrieval of hash value, an entry is inserted into a dictionary in the form of (h_i, i) using h_i, as key. Next step is to perform a pass through the block and compute the hashes and iterative rechecking of the dictionary will be performed based on the matched hashes. It involves two roundtrips: first the client request synchronization of a directory, and then the server replies with some meta information which allows client to decide which files need to updated (Yan et al. 2008), later synchronization is performed in a highly pipelined fashion. Some of the limitations with this approach are –

- High communication cost and low bandwidth savings
- It takes single round of algorithm for remote file synchronization
- Not suited to more complex synchronization scenarios

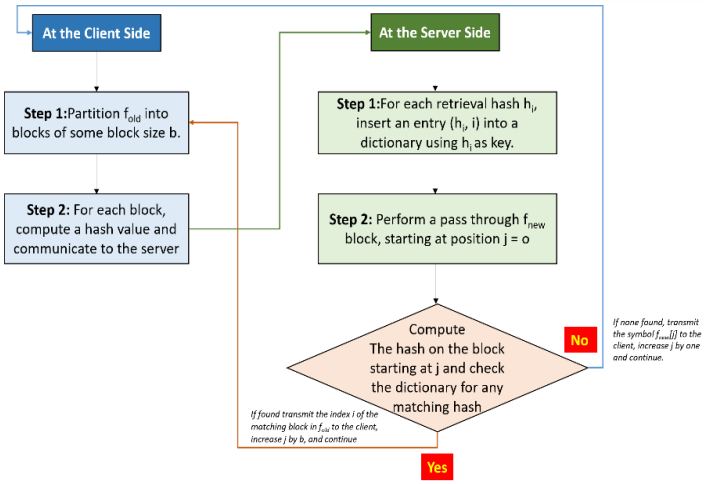

Figure 2. Existing Rsync algorithm with single round for remote file synchronization

Constraint 1: Communication cost and Single Round

Solution: An approach or algorithm based on set reconciliation is proposed where in communication cost is measured in terms of the total number of bits exchanges between the parties. A single round of messages between client and server (Yan et al. 2008) is used, which may also be considered as a limitationIt is assumed that the client has knowledge of the size of symmetric differences between set of hashes on both client and server sides. The modified algorithm is shown in Figure 3. The main idea of the algorithm is that the file versions are partitioned locally into overlapping blocks by using the two-way minimum technique and representing the blocks by their hashes. Then, a set reconciliation method is used to transmit a single message from client to server so that the server knows which of the blocks in f_{new} are already known to the client. Then the server transmits f_{new} to the clients.

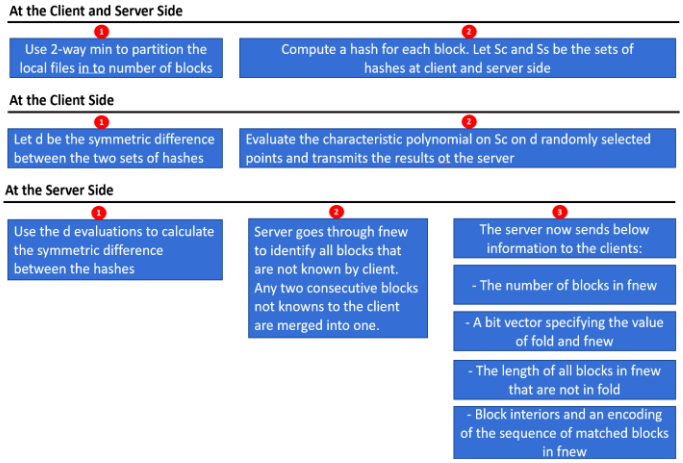

Figure 3. Modified algorithm to the method shown in figure 2. An approach or algorithm based on set reconciliation, where in communication cost is measured in terms of the total number of bits exchanges between the two parties

In above approaches, new algorithms for remote file synchronization that use a single communication round were proposed. Communication costs were lowered assuming that the files are similar. This approach may also provide an alternative to using fixed-size blocks for efficient file synchronization. Another interesting improvement to the existing method could be: *for file synchronization, use multiple round of algorithm instead of single round.*

Constraint 2: Problem with Complex synchronization scenarios

Solution: Rsync is still the most popular tool for synchronizing small datasets in uni- and bi-directional file sync, the file size could be counted in gigabytes. This algorithm provides a uni-directional approach which is appropriate for small data sets across relatively low-latency networks (Konstantin, 2019). Another limitation with this algorithm could be it takes large amount of time to scan a source and target file system for changes and to compare or synchronize the changes over network of varying conditions

(Konstantin, 2019). So, it is not well suited to more complex synchronization scenarios. A method Resilio connect could be a best alternative to Rsync for real-time data synchronization. It scales out data movement in parallel over any network, efficiently scaling transfer performance up to 20 times faster than rsync (Figure 4).

Figure 4. Resilio connect for handling complex synchronization scenerios for large data files

SOLUTION STUDY-III
SPACE MANAGEMENT FOR LARGE FILES

A limitation found in the rsync algorithm, referred to earlier, is the amount of space needed to perform the copying of files, especially with larger files. This condition makes "rsync unusable on mobile devices with limited memories" (Rasch and Burns, 2003). They created a variation of rsync to allow rsync to occur with "in-place reconstruction", an algorithm they termed, ip-rsync. Unlike in rsync, ip-rsync undergoes reordering when being added and copied between the original and remote file. This is accomplished by a destination offset, which is added to each command. The reordering also helps ensure the transfer is corruption-free. In practice, ip-rsync and rsync begin and end the same way. Both generate checksums, which the sender stores in a hash table while they check for files that match. Both also compress the file size, if needed, and tell the system they are finished with the protocol. After the hash table is formed, the two differ. Rasch and Burns protocol adds copy and add commands to the code, which they use to create dependencies between commands. These commands are sorted topologically before being sent to the remote file. This allows the system to read all copy and add commands in order needed based on file layout, instead of when the checksum was matched. The use of a copy command allows the system to release the space that was held by that dependency. Table 2 below shows the difference between rsync and ip-rsync.

Table 2: Rsync vs ip-rsync (Differences are higlighted with blue)

Who	Rsync	Ip-rsync
G	Generate checksums	Generate checksums
G	Send checksums	Send Checksums
S	Build hash table from received checksums	Build hash table from received checksums
S	Scan source file for match	Buffer all copy and add commands
S		Construct dependency graph among copy commands
S		Send sorted copy commands, followed by add to R
R	Apply commands when received then copy data or insert data	Apply commands by first going to received offset then copying data or inserting data
R	Decrease file size if needed	Decrease file size if needed
R	Alert G of file's completion	Alert G of file's completion

In the table, G stands for a generator, S is the sender and R is the receiver.

Ip-rsync experiments show a better overall performance, especially with reduction of disk writes and file system block allocations. This improvement comes from ip-rsync's ability to skip copy commands that have the same source and destination, which are parts that are already the same, while rsync must copy these sections anyways (Rasch and Burns, 2003). Multiple, large files would show a noticeable difference between ip-rsync and rsync. IP-rsync has limitations as well. Because it waits for all checksums to be matched and the copy command to be organized, it cannot perform multiple copies in a single instance. It must go in order of the sorted hash list. Tests also show that ip-rsync uses more bandwidth than regular rsync (Rasch and Burns, 2003). Their new algorithm also does not have error handling, when the article was written. The ip-rsync algorithm is not perfected and more experiments need to be performed to truly analyze performance difference and the best way to optimize the new protocol. Rasch and Burns (2003) intend for this algorithm to be used on devices with more space limitations, such as mobile devices, but acknowledge that this algorithm could be highly successful in high bandwidth situations. Another future experiment will analyze the use of overlapping windows in the ip-sync protocol. Ip-rsync algorithm offers to make improvements to the space limits presented by rsync holding temporary space while processing files. It makes some improvements but still could be improved further, which Rasch and Burns (2003) are currently experimenting with.

REFERENCES

[1] Develop Paper (2019, April 20). *Principle and application of Rsync algorithm | Develop Paper*. Retrieved from http://developpaper.com/principle-and-application-of-rsync-algorithm/

[2] Irmak, U., Mihaylov, S., & Suel, T. (2005, March). Improved single-round protocols for remote file synchronization. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* (Vol. 3, pp. 1665-1676). IEEE.

[3] Konstantin (2019, November 6). Resilio Connect File Sync Software | Connecting Massive Data Flows. *Rsync Alternative: Resilio Connect | Resilio Blog*. Retrieved from http://www.resilio.com/blog/rsync-alternative

[4] Petroni, A., Cuomo, F., Schepis, L., Biagi, M., Listanti, M., & Scarano, G. (2018). Adaptive data synchronization algorithm for IoT-oriented low-power wide-area networks. *Sensors*, *18*(11), 4053.

[5] R1Soft. (2012, April 23). *RSYNC Backup*. Retrieved from R1Soft: http://wiki.r1soft.com/display/TP/rsync+Backup

[6] Rasch, D., & Burns, R. C. (2003, June). In-Place Rsync: File Synchronization for Mobile and Wireless Devices. In *USENIX Annual Technical Conference, FREENIX Track* (Vol. 100).

[7] Suel, T., Noel, P., & Trendafilov, D. (2004, April). Improved file synchronization techniques for maintaining large replicated collections over slow networks. In *Proceedings. 20th International Conference on Data Engineering* (pp. 153-164). IEEE.

[8] Sun, Z., Lv, Z., Wang, H., Li, Z., Jia, F., & Lai, C. (2020). Sensing Cloud Computing in Internet of Things: A Novel Data Scheduling Optimization Algorithm. *IEEE Access*, *8*, 42141-42153.

[9] Tridgell, A., & Mackerras, P. (1996). The rsync algorithm.

[10] Wang, T., Zhou, J., Liu, A., Bhuiyan, M. Z. A., Wang, G., & Jia, W. (2018). Fog-based computing and storage offloading for data synchronization in IoT. *IEEE Internet of Things Journal*, *6*(3), 4272-4282.

[11] Yan, H., Irmak, U., & Suel, T. (2008, April). Algorithms for low-latency remote file synchronization. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications* (pp. 156-160). IEEE.

[12] Zhang, C., & Qi, D. (2020, November). How to parallelize Rsync. In *Journal of Physics: Conference Series* (Vol. 1684, No. 1, p. 012080). IOP Publishing.