Evaluation of Remote Data Compression Methods

Romina DRUTA[1]*, Cristian-Filip DRUTA[2], Ioan SILEA[1]

[1] Automation and Applied Informatics Department, Faculty of Automation and Computer Science, Politehnica University of Timișoara, 300006 Timisoara, Romania
romina.druta@student.upt.ro (**Corresponding author*), ioan.silea@upt.ro

[2] Applied Electronics Department, Faculty of Electronics, Telecommunications and Information Technologies, Politehnica University of Timișoara, 300006 Timisoara, Romania
cristian-filip.druta@student.upt.ro

Abstract: The present era is one of Big Data, digitalization, Internet of Things and Internet of Everything, which imply the daily creation of an enormous amount of useful content with a very high number of producers and consumers for the online information. The ascending trend for Internet data, has made clear the necessity of defining and engineering innovative solutions for coping with redundant transfers, which led to performing smart data transfers for obtaining an increased throughput, data availability and resource utilization and implicitly to a cost reduction and to avoiding bottlenecks and denial of service issues. Internet data employed by an Internet user must be consistent, so distributed systems are gaining research interest with regard to concurrency control, atomic transfers, data replication and synchronization, compression and decompression, correction or other potential problems. Two different versions of a file have a high similarity and as synchronization is concerned, the delta between the second version and the initial version of the file applied to its initial version will provide a better transfer throughput, thus an efficient data deduplication technique is necessary and worth analyzing in order to minimize the cost of synchronization. This paper focuses on optimizing the bandwidth utilization for remote data synchronization, and proposes a prototype based on three classic open-source data compression methods. The experiments carried out show how these compression utilities along with the transfer of data perform the synchronization of large data sets between two remote sites and how the use of compression helps to reduce the data size on storage devices along with decreasing the network bandwidth significantly. The novelty of this paper lies in the fact that it combines two different compression algorithms in order to provide better compression rates.

Keywords: Data replication, Delta encoding, Differential file transfer, Big Data, Network transfer, Rsync.

1. Introduction

According to CISCO VNI by 2022 global IP traffic will triple and the global networked devices and connections will reach 28.5 bilion (CISCO Systems, 2020). Thus a major challenge for the Internet world is the large volume of data which is increasing daily due to the high usage of devices. This problem is also known as the "BIG DATA" problem. The world is "swimming" in data, and the pool is getting deeper at an alarming rate according to IDC which forecasts the Global Datasphere to grow to 175 ZB by 2025 (Reinsel, Gantz & Rydning, 2018).

A crucial problem for organizations with a large number of locations across the globe, is to store the data and to transfer it as fast as possible, in order to make it available everywhere. Smart devices like mobile phones, tablets, smartwatches and others, have a limited amount of storage space and bandwidth. Moreover, the recent trend of cloud computing has changed the way people view IT resources. More and more companies rent storage and computing time from providers like Google Cloud, Microsoft Azure, Amazon Web Services and this has resulted in a more cost-effective use of resources.

Since 1976 researchers have been trying to find optimal models to achieve minimal costs for files storage and transmission, by dynamically assigning the file in a computer network (Segall, 1976; Segall & Sandell, 1979). Several years later in order to achieve network performance and reliability for distributed systems, files migration algorithms were developed and the call for data compression techniques started to be evident (Gavish & Liu, 1990; Shu et al., 2004)

Data replication techniques and strategies have been studied in order to reduce costs and improve performance for the data transferred in the cloud systems (Tos et. al., 2016; George & Edwin, 2017) or IoT based systems (Qaim & Özkasap, 2018). However, often the data that needs to be copied is very similar to an old version of data that already exists on the target site. In this case, time and resources are wasted because copying a new version of data requires copying all of the unchanged data too. Thus data deduplication techniques have started to became highly used in cloud datacenters for big data backups (Bhalerao & Pawar, 2017) or for database systems (Xu et al., 2015, Xu et al., 2017).

Given this context, reducing both the size of the data and the transfer time remain big challenges. Another way to resolve these problems is to transfer only the difference between an old version

https://doi.org/10.24846/v31i1y202206

and a new version of the data, between the sites, thereby reducing the transfer time, the data size on the disk, and optimizing the bandwidth.

The purpose of this paper is to describe and analyze a technique for the replication and storage optimization with regard to the big data sets of files, an activity which implies the analysis of different types of files, and delta encoding of these files, using three well-known applications: Rsync (Tridgell, 1999), Xdelta3 (MacDonald, 2000), Bsdiff/Bspatch (Naïve, 2013). A structural analysis of the mentioned delta encoding algorithms is performed in order to determine, which delta encoder generates the smallest delta between two versions of a data set.

This article is structured as follows. Section 2 presents the main concepts related to data replication, focusing on delta encoding and remote file synchronization. Section 3 outlines the different binary diff tools and their usage, while Section 4 describes the methods used for the synchronization between two versions of a software project. Section 5 introduces the results of the tests along with an analysis of the problems and limitations encountered and Section 6 presents the conclusion of this paper.

2. Background

In Computer Science, data replication (Charron-Bost, Pedone & Schiper, 2010) is the process of storing the data on multiple storage devices. One of the main goals of this process is to increase the availability of resources. Given the large amounts of data that is generated every day, replicating this data is becoming more and more challenging.

The simplest method to replicate data is to copy all of it from one site to another. But if the data that it is to be transferred has been only partially changed, another way to replicate the content on the target site is to maintain an old copy on the target and synchronize the files and the tree directories on both sides.

Delta encoding (Suel, 2018) is a way of storing or transmitting data in the form of differences between sequential data rather than complete files; more generally this is known as data differencing (or delta compression). The delta compression can be explained as follows: host C has a copy of *fold* and host S has copies of both *fnew* and *fold*, where *fold* is the old file and *fnew* is the new file. The goal for host S is to compute a file *fdelta*, which is the delta difference file, of minimum size such that host C could reconstruct *fnew* from *fold*. If the delta between the files from two distant sites is computed, one can speak of remote differential compression.

Remote file synchronization is the process of adding, changing, or deleting a file in one location, and having the same file added, changed, or deleted at another remote location. For example, as it can be seen in Figure 1, if the Remote Server has a copy of the file fold and the Local Server has a copy of both fold and fnew, and the aim is to generate fnew on the Remote Server, one needs a protocol that minimizes the communication cost between the two computers.

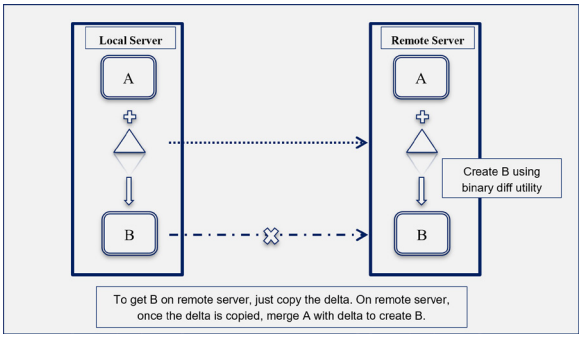

Figure 1. File replication using delta encoding

One method of synchronizing the files is to compute the delta between the old version and new version, while transferring only the differences. To do this one can use different methods based on remote delta encoding algorithms. Remote differential compression (Suel & Memon, 2002) (remote delta encoding) is an algorithm that allows two files to have their content synchronized by communicating only the difference between them, in a client server configuration.

3. Methods and Related Work

This section aims to present the most common binary diff generation algorithms and tools used as solutions for the file synchronization process and other methods used for files synchronization.

In computing, a diff tool is a file comparison utility that outputs the differences between two files. Recent versions, which offer performance improvements and compression, also support binary files. Delta compression tools such as Xdelta3, Bsdiff/Bspatch and Zdelta (Trendafilov, Memon & Suel, 2002) are generally used

for generating differences between two files (the old and the new version) if both files are available on the same machine. Remote delta compression tools (Shiala, Majhib & Phatakc, 2015), like Rsync are used for synchronizing files between remote hosts (a source site which has a newer version, and a target site which retains the old version), by producing a differential file which will be then transferred and applied on the target site. However, Rsync can be used for synchronizing files that are on the same machine. Rsync is among the most popular tools for remote delta encoding. Nonetheless, a problem which appears when using this method is the huge amount of time taken to transfer small chunks of data between two sites separated by a WAN link, the latency on the network being a factor in this process.

Recently, research studies have been performed in order to find solutions for web-based synchronization methods like WebDeltaSync (Xiao et al., 2018), or for mobile cloud storage services QuickSync (Cui et al., 2015). Other studies have been focusing on hybrid synchronization like PandaSync, which proposes a method which involves full synchronization and delta synchronization depending on the network performance (Wu et al., 2019). A presentation of Xdelta3, Bsdiff and Courgette (ProgrammerSought, 2021) shows that Bsdiff generates 50% fewer binary patches and Courgette has an even better compression rate, but this analysis does not take into account the Rsync algorithm which is still widely employed for file updates and migration.

Rsync

Rsync is the most well-known tool for file synchronization between two machines across the network. What makes Rsync different from other file transfer tools is that it detects the differences between the source data set and target data set and then transmits these across the network, so that the target can apply them, resulting in a complete file synchronization between the two hosts. Rsync is a tool that integrates multiple functionalities including: an algorithm which detects differences between two files, a network transport protocol and a data compression algorithm.

As previously noted, Rsync is employed in order to determine changes when comparing two versions of one file without having both versions of the file on the same machine.

The network protocol used by Rsync can be summarized as follows: on the computer where the old versions of the files, are hosted, Rsync calculates the checksums for each block of data. These checksums are then sent to the computer which hosts the new version of files. After detecting the changes between blocks, the computer that hosts the new version sends instructions on how to create the copy of the new version to the computer that has the old data. The instructions include the actual new data and the references on blocks that hadn't been changed in the new file.

Rsync is used for synchronizing files in distributed client-server applications (Faiz & Shanker, 2016), backup and restore on e-learning platforms (Purnama et. al., 2016) and updates in sensor networks (Qaim & Özkasap, 2018). Other recent studies show that Rsync can be used in emergency communication systems (Schepis et. al., 2019), cloud based IoT applications (Kolano, 2015; Sánchez-Gallegos et al., 2020) and fog computing platforms (Puliafito et al., 2020; Puliafito et al., 2021).

Xdelta

Developed by Josh MacDonald, Xdelta3 is a set of tools for reading and writing compressed deltas. The algorithm, first used in Xdelta1, was based on the Rsync algorithm; the main difference is that the size of the blocks was smaller than the block size that Rsync uses. The new version, Xdelta3, uses the VCDIFF (Korn et al., 2002) encoding library. The VCDIFF Generic Differencing and Compression Data format supports better compression and it allows the data to be easily transported among computers. The synchronization of files is optimized for high speed while generating smaller deltas. Both files involved in the synchronization process must be available locally. Xdelta3 was used in malware analysis by Fowler (2017), and for the study of a VM handoff in edge computing (Ha et al., 2017).

Bsdiff/Bspatch

Bsdiff and Bspatch are tools for building and applying patches to binary files. These

compression utilities are similar to Xdelta3 in that they do not provide network transport functionality, and can produce a delta file even smaller than the ones produced by Rsync or Xdelta3. The algorithm used by Bsdiff/Bspatch functions by reading the old file and indexing the content. After that, the new file is read in order to find the blocks that match exactly the blocks of the old file. Unlike other binary tools which put all these changes into a single patch file, Bsdiff builds a patch set which contains 3 parts: a control file, containing ADD and INSERT instructions, a difference file containing the bytewise differences of the approximate matches, and an extra file, containing the bytes which were not part of an approximate match.

One important thing to note about Bsdiff is that it uses an algorithm highly demanding in memory resources. It requires max(17*n,9*n+m)+O(1) bytes of memory, where n is the size of the old file and m is the size of the new file. This solution is not the most appropriate to use because it needs huge resources of memory. Bsdiff was mainly analysed for incremental backups for sensors and resource-constrained microcontrollers (Teraoka, Nakahara & Kurosawa, 2016; Onuma et al., 2016; Stolikj et al., 2013).

4. Proposed Work

The approach presented in this paper aims to separate the remote delta encoding process from the transfer process by computing locally the delta difference between the files and then transferring them by using other compression utilities over the network.

Thus the process of synchronization that we have been using, consists of three stages: a preprocessing stage (obtaining the delta files for two versions of a file which are on the same machine), a transfer stage of the delta data and a "postprocessing" stage which relates to applying the delta files to the old version of the selected files to synchronize them. For the stages of preprocessing and postprocessing, after a careful analysis of available delta generation tools, three different utilities were determined to be worth further evaluation in this paper. In order to describe the steps for remote synchronization using delta differencing before transferring the files, the remote synchronization for Rsync is analysed (see Figure 2).

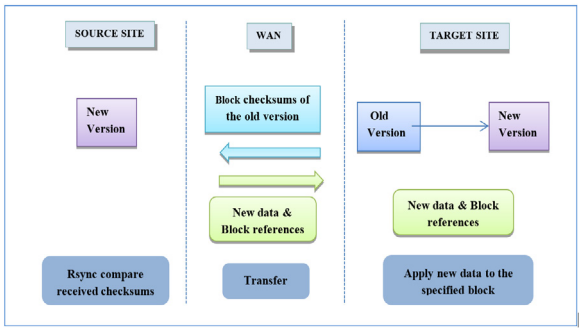

Figure 2. Rsync Remote Synchronization

As previously noted, Rsync is usually used for synchronizing remote hosts, but it can also be employed for computing the difference between two versions of a file in local mode. When Rsync is used between a SOURCE SITE and a TARGET SITE, the TARGET SITE has to send all the block checksums for its version to the SOURCE SITE. After comparing the block checksums with regard to the values received and the values of the block checksums for the new version, the SOURCE SITE will communicate the new data that needs to be updated and the references of the blocks where the data needs to be applied.

Thus, in this case, the bandwidth utilization is greater than if the delta differences were sent after a local delta encoding run between two different versions of a tree directory structure.

As it is shown in Figure 3, the prototype that we have used to synchronize two remote versions includes three steps: the preprocessing step which supplies the delta, the transfer step for transferring the delta over the network to the target site and the final step, namely the postprocessing which consists in applying the delta to the TARGET SITE.

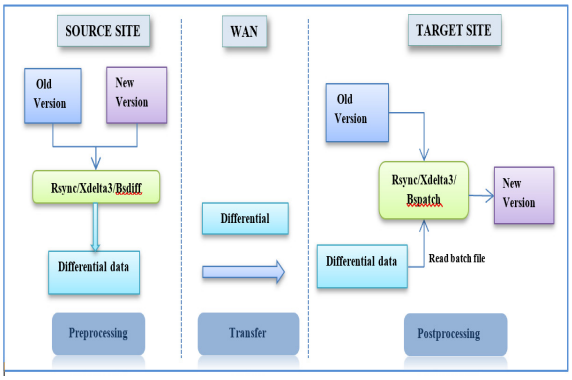

Figure 3. Remote Synchronization using local delta encoding

For each method, the steps used for obtaining the delta difference file set are explained below; it is also explained how this delta data was transferred and finally which were the steps applied at the destination in order to obtain the new version of the data set.

Preprocessing

1. Rsync

To obtain all the differences between two versions in one single file, Rsync was used in -only-write-batch mode. This function allows writing all changes to a single file, which can then be sent and applied on the TARGET site (see Figure 4).

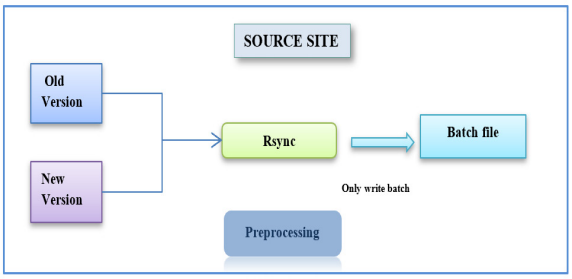

Figure 4. Rsync Preprocessing

The syntax used was:

*Rsync -av --delete -i -8 --progress --stats –only-write-batch=batch_delta_file --recursive --log-file=test_log --out-format=”**** file_size: %l ---- time: %t ---- file_name: %f ---- delta_size: %b ---- itemize_change: %i *****” **NEW_VERSION OLD_VERSION***

2. Xdelta3

For Xdelta3, the preprocessing stage as it can be seen in Figure 5 included the following parts:

a. Running Rsync and processing the output of the process in real time:

- Find which files need to be deleted;
- Find which files are new in the new version;
- Find which files need to be updated.

b. For each delta file that needs to be updated, Xdelta3 was run against the old version and the new version of that file, keeping the generated delta file in a separate directory.

The syntax used for Xdelta3 in encoding mode is the following:

Xdelta3 -A -e –s old_file_version new_file_version delta_file
where:
-e – compress
-s – source file to copy from
- A – provide application header

c. When the directory reaches 100 delta files it is added to a .tar archive, and a new directory will be created for the next 100 delta files

d. All the information related to the files that need to be deleted is stored in a log file

e. All the new files are copied in a directory, and their name and path are written to another log file, in order to be applied on the TARGET site in the postprocessing stage

f. In addition, there is another log file which keeps the information about the delta file, the path and name of the file which was changed when comparing its versions

All the log files, as well as the directory which contains the new files, are added to archive; the final archive file will be sent across the network to be processed on the TARGET site.

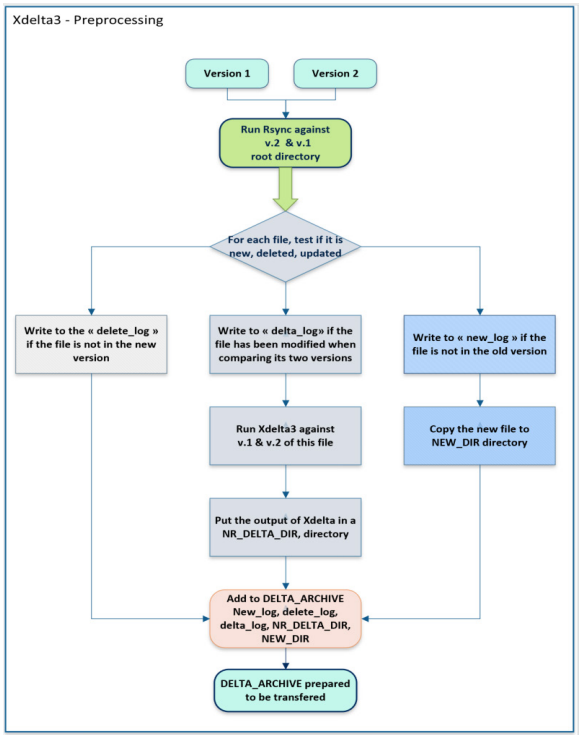

Figure 5. Flow diagram for Xdelta 3 preprocessing using Rsync

3. Bsdiff/Bspatch

The preprocessing stage for Bsdiff/Bspatch is the same as for Xdelta3 the only difference being that Bspatch is called against two versions of the file instead of calling Xdelta3 in mode encoding.

```
Bsdiff old_version new_version delta_file
```

Transfer

To create the new data set files on the remote target site, it is necessary to transfer the delta file in order to apply it to the old version. In order to do this, the time of data transfer was evaluated by using two transfer solutions: Rsync, and BBCP which are open source tools. Rsync can be used as a transfer tool also, being able to transfer the small differences between the files across the WAN.

BBCP (Hanushevsky, 2012) a point-to-point network file copy application, is an alternative to GridFTP (Allcock et al., 2005) for transferring large amounts of data. This utility breaks up the transfer into multiple streams, thereby transferring data much faster than single-streaming utilities such as SCP (Secure Copy Protocol) and SFTP (Secure File Transfer Protocol).

BBCP was employed in order to transfer the batch file obtained after using Rsync, and the .tar archive when using Xdelta3 and Bsdiff.

Postprocessing

Once the delta files arrived at the target, the delta was applied to the old data set files using the decoding option for the software that was used at the source site to encode the differences between the two versions of the respective files. At this step the priority is to find the time taken by each binary diff tool when applying the delta file to the target site. Below, it is described for each binary tool, which were the steps applied in order to obtain the new version of the data set on the target site.

1. Rsync

To apply the delta modifications to the old version, Rsync was used in read-batch file mode, as it is shown in Figure 6.

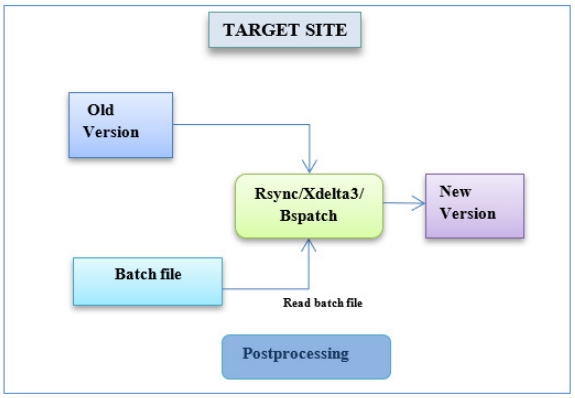

Figure 6. Rsync postprocessing

The syntax used was:

```
Rsync -v --read-batch=*_batch_bin_batch_file
-v – to display the output of Rsync
--read-batch – apply changes stored in the previously
generated file  *_batch_bin_batch_file"
```

2. Xdelta3

Knowing that the delta files previously generated are all contained in a .tar file, the process for Xdelta3 applying patch includes the following steps, which are also described by the process flow from Figure 7:

a Create a copy of the old version to the new version directory, using hard links
b Unpacking the log files, the directories; containing the delta files and the directory which contains the new files that are not at the target site;
c Loading into RAM the information from the log files, including which files need to be deleted, the new files to be created and the delta files;
d Delete the files that are not in the new version and were in the old one from the hard-linked copy directory, using the log which contains information regarding the files that need to be deleted from the new version;
e Copy new files to the hard-linked copy target directory, using the log file which contains the path and the name of the file that needs to be copied;
f Apply delta files, using Xdelta3 decoder for each file that has been modified in the new version. This step is done based on the log file which, for each file, lists the path of the file and the name of the delta file which corresponds to the modified file.

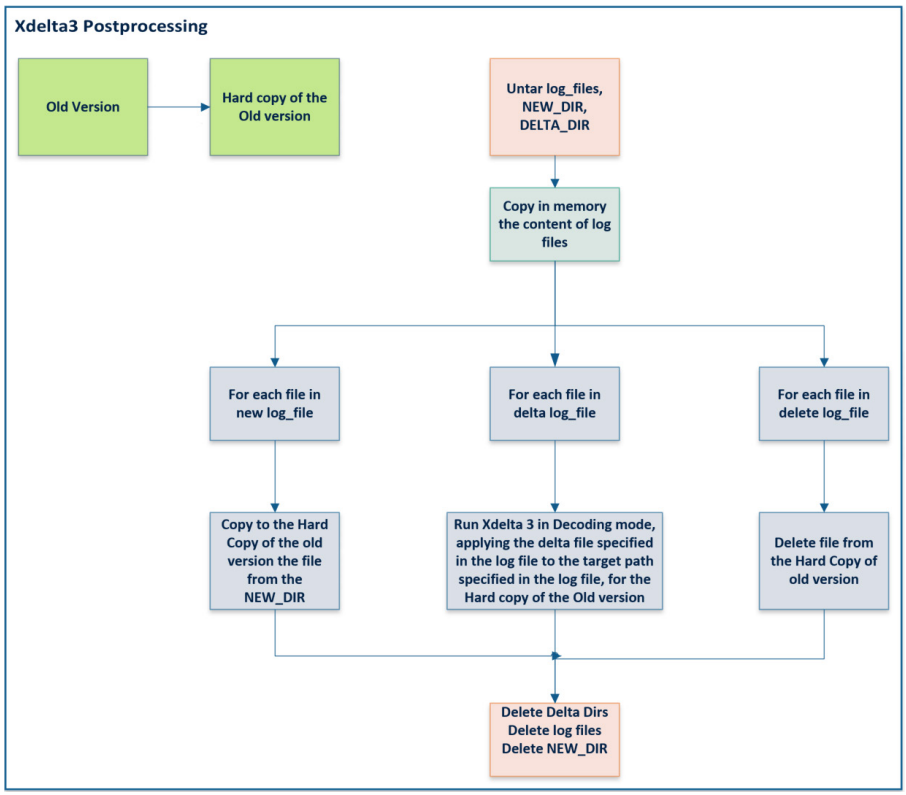

Figure 7. Flow diagram for Xdelta3 postprocessing diagram

The syntax used was:

Xdelta3 –d –s old_file_version delta_file new_file_version
Where
-d – decoding, decompressing

3. Bsdiff/Bspatch

The postprocessing stage for Bsdiff/Bspatch is almost the same as for Xdelta3, the only difference being that Bspatch is run for decoding instead of running Xdelta3 in decoding mode.

The syntax used was:

Bspatch old_file_version delta_file new_file_version
Where
-d – decoding, decompressing

5. Experiments and Results

For the presented experiments four data sets of files were used, which are represented by a software project compiled on Windows and Linux platforms. The tests were performed using two Red Hat Enterprise Linux Servers, one located in North America(Canada), and the other one in Central Europe (Germany), both connected on the same WAN (Wide Area Network) and working at an average speed of 250 Mbps.

In order to achieve a performance comparable to that of Rsync, parallel processes were used both for the preprocessing and postprocessing stages. The testing conditions were the same for all algorithms and there were no other processes running on the servers at the time of these tests. The time-related results were obtained using the *time* command from the Unix system. Consistent results were obtained by carrying out tests repeatedly for three times, small variations of 0.03% were observed.

The results of the experiments carried out will be presented as follows: first the spectrum of data set files will be described, then for the given data sets the size of the selected files will be analyzed within each data set. Afterwards the focus will be on the delta size of those files obtained for a new and an old version by using the three algorithms mentioned above. Finally, the analysis will focus on the preprocessing time in obtaining the delta difference file, the time to transfer the delta file and the time taken by the postprocessing step at the target destination.

Spectrum of Data Set

In this section the type of files that are within a data set are first analyzed. Then, the size of those files and the size of delta files will be analyzed for two versions of a data set. For the proposed experiments let us consider Data Set A and Data Set B which contain the files of a software project compiled on win64_x64 and win32_x86 platforms, and Data Set C and Data Set D with files resulting from the compilation of the project on linux_x86 and linux_x64 platforms.

1. Type of files

In order to analyze the performance of the delta encoding tools depending on the type of the files within a data set, the types of files by artifact (directory) are presented, as it can be seen in Figure 8.

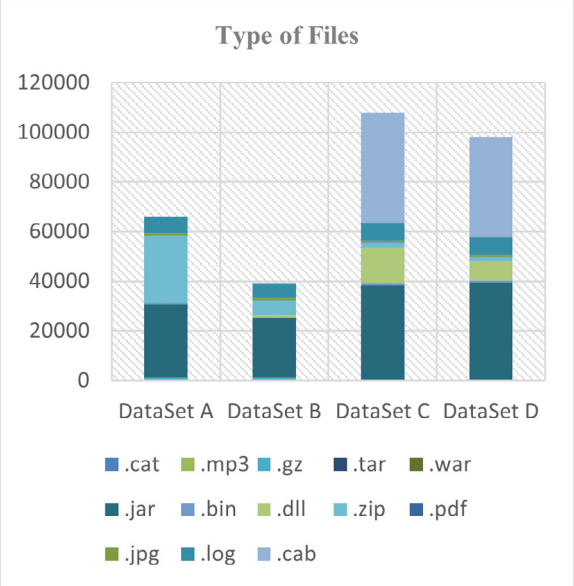

Figure 8. Types of files within an artifact

2. Size of files

At this point, the spectrum of the data set files was analyzed. Thus, one obtained the total number of files and directories, the maximum depth for the directories, and the number of files divided into ranges by their size (e.g. files less than 1kB, files between 1kB and 10 KB, etc…). As it can be seen in Figure 9, most of the files of the data set are smaller than 1kB or they have a size between 1 kB and 10 kB.

3. Size of delta files

In this section, the spectrum of the delta differences between two versions of each data set was analyzed before the compressing algorithms were applied. As Figure 10 shows, it can be stated that 90% of the file differences for two versions of a data set are smaller than 1kB. Sending all of these small differences across the network consumes a large amount of time and bandwidth.

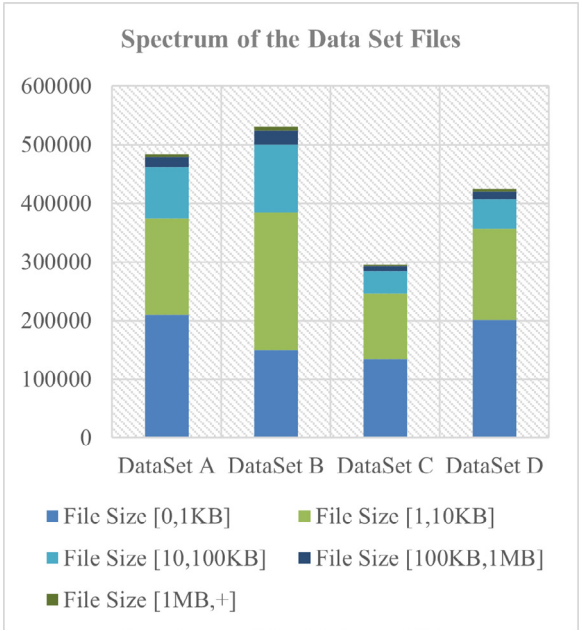

Figure 9. Spectrum of the Data Set Files

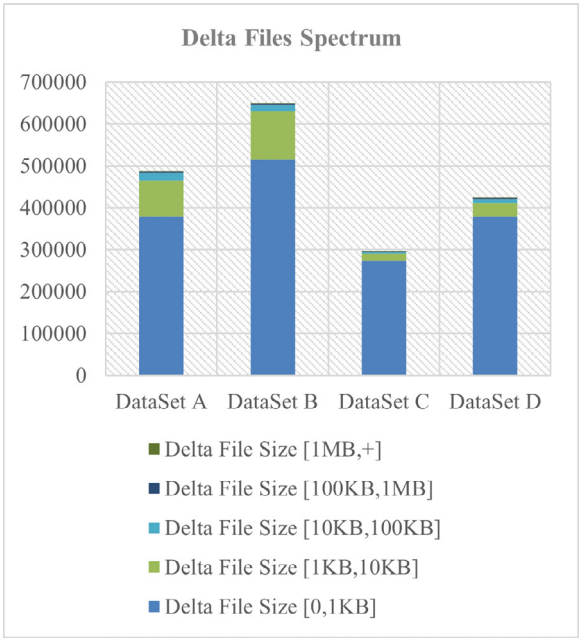

Figure 10. Delta Files Spectrum

When using Rsync in remote mode, these differences must be individually transmitted over

the network. However, creating archive files which contain the differences results in faster transfer rates because they can be read contiguously by the transfer software. This also provides the ability to buffer the data that arrives on the target site in memory and then flush it to disk at a later time, which provides even better transfer rates.

Size of Delta Files for the Compression Algorithms

In this paragraph one analyzes the size of the delta files that are generated by the three delta encoding algorithms described in Section 4.

Looking at the size of the Delta File generated by these utilities, in Figure 11 it can be seen that Rsync is not very efficient when focusing on the size of the delta difference file. The size for the file created by Xdelta3 and Bsdiff/Bspatch is almost the same for the Data Set A and Data Set B. One thing that must be mentioned is that, based on Figure 11, Xdelta3 has a file size smaller than Rsync's batch file if .bin files are compared (Data Set C and Data Set D).

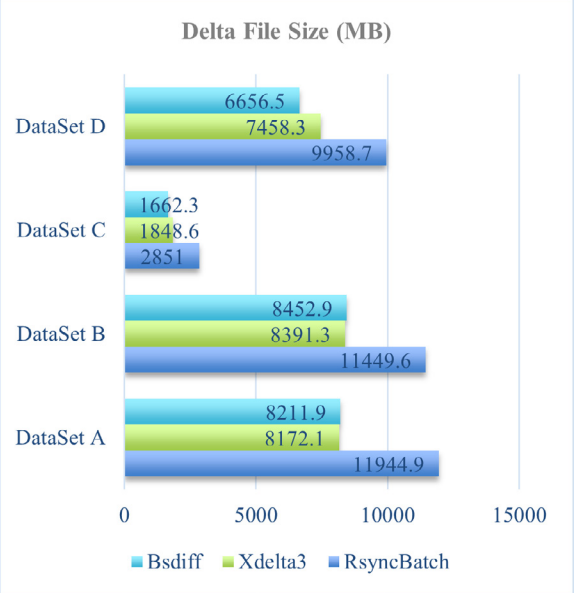

Figure 11. Average of delta file size by DataSet

When comparing Xdelta3 and Bsdiff, the size of delta file in two cases is smaller for Bsdiff than for Xdelta3, but overall the compression performance is the same when talking about the file size obtained from the encoding process.

Figure 12 illustrates the total size of two versions A and B which contain all the files of the four datasets analyzed in Section 5.

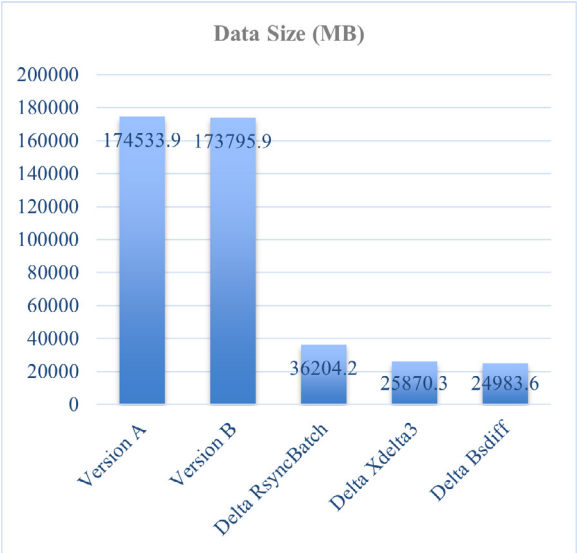

Figure 12. Total size of the entire version compared to the total size of delta file

The size of these versions is compared with the size of the generated delta files obtained through the three utilities described in Section 4. One can imagine that if it was necessary to transfer the entire version of a project, it would have taken much longer than transferring only the delta encoding files.

Preprocessing time

Figure 13 illustrates the preprocessing time obtained by the delta encoding algorithms, showing that Bsdiff takes the highest time to compute the delta difference.

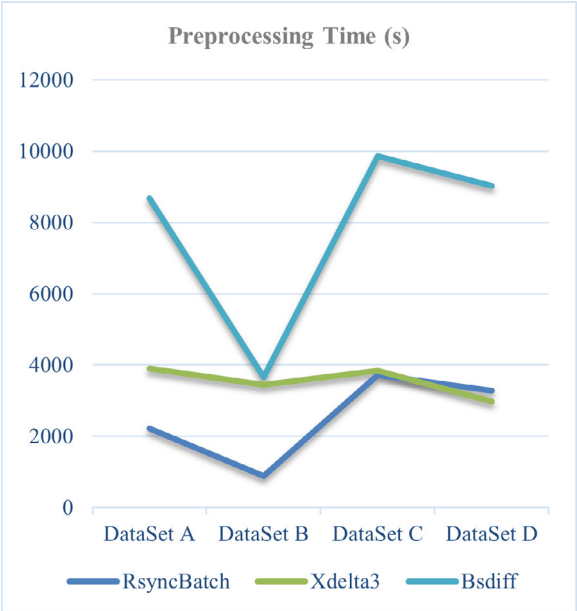

Figure 13. Preprocessing time by artifact

If one compares only Rsync and Xdelta3, it can be stated that Xdelta3 takes more time than Rsync also, but the difference is not as significant as in the previous case. For the Data Set C and Data Set D which include .cab and .bin files (see Figure 8) the performance of Xdelta3 is similar to that of Rsync, in comparison with the Data Set A and Data Set B where a lot of .zip files are present.

Transfer time

It is known that the smaller a file is, the faster the transfer will be. In this case, it can be stated that transferring the .tar archive created using the Xdelta3 method is faster than transferring the batch file created with Rsync, as the results show in Figure 14.

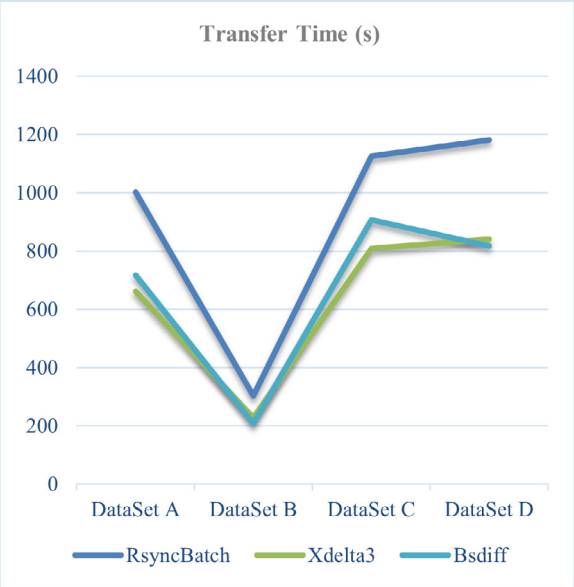

Figure 14. Transfer time for the delta files

Postprocessing time

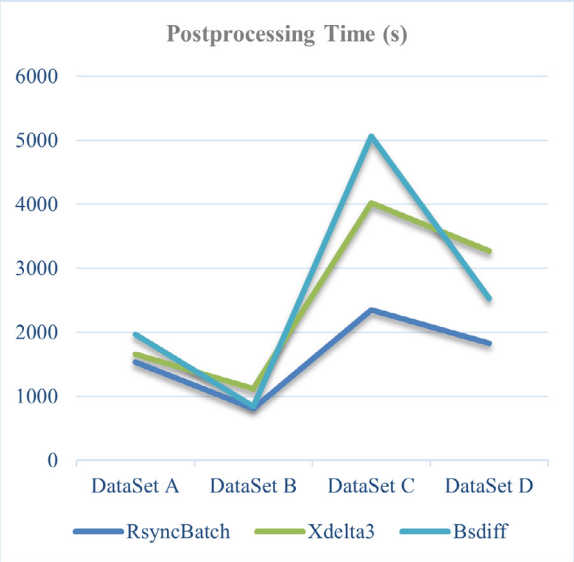

Figure 15. Postprocessing time by artifact

At the stage of postprocessing where the delta differences are applied on the target site, in Figure 15 it can be seen that Rsync performs the fastest out of all three methods. It can be also noticed that Xdelta3 can be slower than Bsdiff/Bspatch at a certain degree.

Total Synchronization time

For the total synchronization time, as it can be seen in Figure 16, Rsync used in batch mode performs the best. When trying to apply remote synchronization for two versions using Rsync, the small-size files transmitted over the network lead to a higher synchronization time. Xdelta3 performs better than Bsdiff/Bspatch but it has a higher synchronization time in comparison with Rsync used in batch mode.

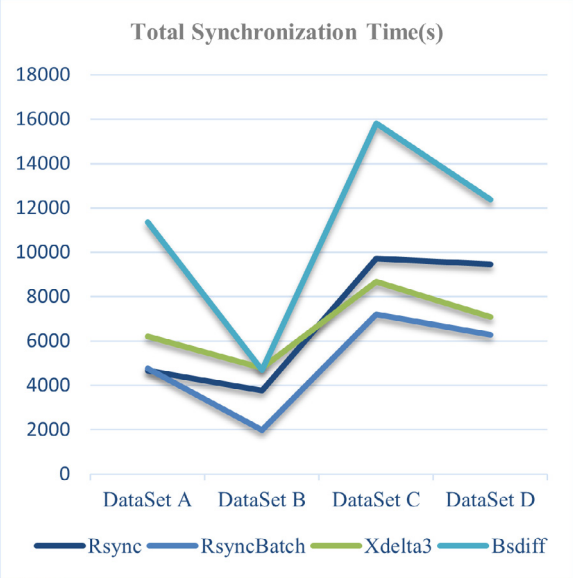

Figure 16. Total processing time by artifact

6. Conclusion

This paper analyzed the results obtained for the remote delta encoding process, using Rsync, Xdelta3 and Bsdiff. These utilities were chosen because they are open-source and widely used.

As it can be seen in Figure 12, for a total size of the data sets of almost 175 GB, the total size of delta files obtained for the two proposed versions is higher than 36 GB when using Rsync. Xdelta3 has a better compression rate but the time required to process the data is significantly higher than in the case of Rsync, even if parallel processing is used. Bsdiff/Bspatch obtained comparable results with those of Xdelta3, being faster than it at the

postprocessing stage (Figure 15), but it consumes a lot of resources.

It can be concluded that Rsync is the fastest tool for performing the synchronization, but it consumes a lot of bandwidth to transfer the delta difference file. Xdelta3 is not as fast as Rsync, but the delta files are almost two times smaller than Rsync's batch files, which leads to a lower bandwidth consumption.

As a result of this research, it can be stated that it is advisable to use Rsync as a compression algorithm when the cost of the bandwidth is not a problem, otherwise it is more favorable to use Xdelta3 or another algorithm with a good compression rate and another utility for transferring the delta difference across the network.

Based on the obtained results, the future aim would be to build an optimized transfer system for security and remote collaboration.

REFERENCES

Allcock, W., Bresnahan, J., Kettimuthu, R. & Link, M. (2005). The Globus Striped GridFTP Framework and Server. In *ACM/IEEE SC 2005 Conference (SC'05)*, (p. 54). DOI:10.1109/SC.2005.72

Bhalerao, A. & Pawar, A. (2017). A survey: On data deduplication for efficiently utilizing cloud storage for big data backups. In *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, Tirunelveli (pp. 933-938).

Charron-Bost, B., Pedone, F. & Schiper, A. (2010). *Replication, Theory and Practice*, Springer.

Cisco Systems (2020). *Cisco Annual Internet Report (2018-2023)*, White Paper. Available at: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>, last accessed: 27 of January, 2022.

Cui, Y., Lai, Z., Wang, X. & Dai, N. (2015). QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15)*, Paris, France (pp. 3513-3526).

Faiz, M. & Shanker, U. (2016). Data synchronization in distributed client-server applications. In *2016 IEEE International Conference on Engineering and Technology (ICETECH),* Coimbatore (pp. 611-616).

Fowler, J. E. (2017). Compression of Virtual-Machine Memory in Dynamic Malware Analysis, *Journal of Digital Forensics, Security and Law*, *12*(1). Available at: <https://commons.erau.edu/jdfsl/vol12/iss1/9>. DOI: 10.15394/jdfsl.2017.1437

Gavish, B. & Liu, S. O. R. (1990). Dynamic file migration in distributed computer systems, *Communications of the ACM, 33*(2), 177-189. DOI: 10.1145/75577.75583

George, S. & Edwin, E. B. (2017). A Review on Data Replication Strategy in Cloud Computing. In *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC),* Coimbatore (pp. 1-4).

Ha, K., Abe, Y., Eiszler, T., Chen, Z., Hu, W., Amos, B., Upadhyaya, R., Pillai, P. & Satyanarayanan, M. (2017). You can teach elephants to dance: agile VM handoff for edge computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '17). Association for Computing Machinery*, New York, NY, USA (pp. 1-14). DOI: 10.1145/3132211.3134453

Hanushevsky, A. (2012). *BBCP*. Available at: <https://www.slac.stanford.edu/~abh/bbcp/>, last accessed: 27 of January, 2022.

Kolano, P. Z. (2015). Automatically encapsulating HPC best practices into data transfers. In *Proceedings of the Second International Workshop on HPC User Support Tools (HUST'15),* Association for Computing Machinery, New York, NY, USA, Article 1 (pp. 1-12). DOI: 10.1145/2834996.2834997

Korn, D. G., MacDonald, J., Mogul, J. C. & Vo, K. (2002). The VCDIFF Generic Differencing and Compression Data Format. In *Internet Engineering Task Force (IETF), Network Working Group, RFC, 3284*. Available at: <https://tools.ietf.org/html/rfc3284>.

MacDonald, J. (2000). *File system support for delta compression*, MS Thesis, UC Berkeley. Available at: <http://xdelta.org/>.

Naïve, C. (2003). *Differences of Executable Code*, Computing Lab, Oxford University. Available at: <http://www.daemonology.net/Bsdiff/>.

Onuma, Y., Nozawa, M., Terashima, Y. & Kiyohara, R. (2016). Improved Software Updating for Automotive ECUs: Code Compression. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Atlanta, GA (pp. 319-324).

Programmer Sought (n. d.). *Comparison of three differential algorithms for Xdelta3 bsdiff Courgette*. Available at: <https://www.programmersought.com/article/479736363/>, last accessed: 18 of August, 2021.

Puliafito, C., Gonçalves, D. M., Lopes, M. M., Martins, L. L., Madeira, E., Mingozzi, E., Rana, O. & Bittencourt, L. F. (2020). MobFogSim: Simulation of mobility and migration for fog computing, *Simulation Modelling Practice and Theory*, *101*, 102062. DOI: 10.1016/j.simpat.2019.102062

Puliafito, C., Vallati, C., Mingozzi, E., Merlino, G. & Longo, F. (2021). Design and evaluation of a fog

platform supporting device mobility through container migration, *Pervasive and Mobile Computing*, *74*, 101415. DOI: 10.1016/j.pmcj.2021.101415

Purnama, F., Usagawa, T., Ijtihadie, R. M. & Linawati (2016). Rsync and Rdiff implementation on Moodle's backup and restore feature for course synchronization over the network. In *2016 IEEE Region 10 Symposium (TENSYMP)*, Bali (pp. 24-29).

Qaim, W. B. & Özkasap, Ö.(2018). State-of-the-Art Data Replication Techniques in IoT-Based Sensor Systems. In *2018 IEEE Globecom Workshops (GC Wkshps),* Abu Dhabi, United Arab Emirates (pp. 1-6).

Reinsel, D., Gantz, J. & Rydning, J. (2018). *The Digitization of the World from Edge to Core*, IDC White Paper sponsored by Seagate. Available at: <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>.

Sánchez-Gallegos, D. D., Carrizales-Espinoza, D., Reyes-Anastacio, H. G., Gonzalez-Compean, J. L., Carretero, J., Morales-Sandoval, M. & Galaviz-Mosqueda, A. (2020). From the edge to the cloud: A continuous delivery and preparation model for processing big IoT data, *Simulation Modelling Practice and Theory, 105,* 102136. DOI: 10.1016/j.simpat.2020.102136

Schepis, L., Cuomo, F., Petroni, A., Biagi, M., Listanti, M. & Scarano, G. (2019). Adaptive Data Update for Cloud-based Internet of Things applications. In *Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era (PERSIST-IoT'19)*, Association for Computing Machinery, New York, NY, USA (pp. 13-18). DOI: 10.1145/3331052.3332472

Segall, A. (1976). Dynamic file assignment in a computer network, *IEEE Transactions on Automatic Control, 21*(2), 161-173. DOI: 10.1109/TAC.1976.1101193

Segall, A. & Sandell, N. (1979). Dynamic file assignment in a computer network–Part II: Decentralized control, *IEEE Transactions on Automatic Control, 24*(5), 709-716. DOI: 10.1109/TAC.1979.1102169

Shiala, G., Majhib, S. K. & Phatakc, D. B. (2015). A Comparison Study for File Synchronisation. In *International Conference on Intelligent Computing, Communication & Convergence (ICC-2015),* (pp. 133-141).

Shu, J., Wang, B., Meng, W. & Deng, Y. (2004). Policy of file migration at server in cluster file system. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004),* (pp. 691-698). DOI: 10.1109/CCGrid.2004.1336700

Stolikj, M., Cuijpers, P. J. L. & Lukkien, J. J. (2013). Efficient reprogramming of wireless sensor networks using incremental updates. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),* San Diego, CA (pp. 584-589).

Suel, T. (2018). Delta Compression Techniques. In Sakr S. & Zomaya, A. (eds.), *Encyclopedia of Big Data Technologies,* Springer.

Suel, T. & Memon, N. (2002). Algorithms for delta compression and remote file synchronization. In Sayood S. (ed.), *Handbook of Lossless Compression*, Academic Press.

Teraoka, H., Nakahara, F. & Kurosawa, K. (2016). Incremental update method for resource-constrained in-vehicle ECUs. In *2016 IEEE 5th Global Conference on Consumer Electronics*, Kyoto (pp. 1-2).

Tos, U., Mokadem, R., Hameurlain, A., Ayav, T. & Bora, S. (2016). A Performance and Profit Oriented Data Replication Strategy for Cloud Systems. In *2016 International IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress* (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse (pp. 780-787).

Trendafilov, D., Memon, N. & Suel, T. (2002). *Zdelta: a simple delta compression tool*, Technical Report, Polytechnic University, CIS Department. Available at: <http://cis.poly.edu/zdelta/>.

Tridgell, A. (1999). *Efficient Algorithms for Sorting and Synchronization*, Ph.D. Thesis, The Australian National University. Available at: <https://rsync.samba.org/documentation.html>.

Wu, S., Liu, L., Jiang, H., Che, H. & Mao, B. (2019). PandaSync: Network and Workload Aware Hybrid Cloud Sync Optimization. In *Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS),* Dallas, Texas, United States (pp. 282-292).

Xiao, H., Li, Z., Zhai, E., Xu, T., Li, Y., Liu, Y. & Zhang, Q. (2018). Towards Web-based Delta Synchronization for Cloud Storage Services. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18),* Oakland, CA (pp. 155-168).

Xie, F., Yan, J. & Shen, J. (2017). Towards Cost Reduction in Cloud-Based Workflow Management through Data Replication. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD),* Shanghai (pp. 94-99).

Xu, L., Pavlo, A., Sengupta, S. & Gagner, G. R. (2017). Online Deduplication for Databases. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17),* Association for Computing Machinery, New York, NY, USA (pp. 1355-1368). DOI: 10.1145/3035918.3035938

Xu, L., Pavlo, A., Sengupta, S., Li, J. & Gagner, G. R. (2015). Reducing replication bandwidth for distributed document databases. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15). Association for Computing Machinery*, New York, NY, USA (pp. 222-235). DOI: 10.1145/2806777.2806840