International Journal of Innovative Technology and Interdisciplinary Sciences
www.IJITIS.org
ISSN:2613-7305
Volume 5, Issue 3, pp. 1006-1013, 2022
DOI: https://doi.org/10.15157/IJITIS.2022.5.3.1006-1013
Received June 11, 2022; Accepted September 30, 2022

Using Concurrency and File Integrity Verification to Transfer Files from Client to Server

Aleksander Biberaj [*a1], **Alban Rakipi** [b1], **Bledar Kazia** [c2], **Esmeralda Kazia** [d2], **Sonila Xhaferllari** [e1]

[1]Department of Electronics and Telecommunications
Polytechnic University of Tirana, Tirana, Albania

[2]Department of Computer Engineering,
Canadian Institute of Technology, Tirana, Albania

*[a]abiberaj@fti.edu.al; [b]arakipi@fti.edu.al; [c]bledi.kazia@cit.edu.al;
[d]esmeralda.kazia@cit.edu.al; [e]sxhaferllari@fti.edu.al;

ABSTRACT

In this research work it has been provided a method for ensuring the integrity of the data throughout their transfer from the client to the server by computing the checksum of the data on the server and comparing it with the checksum of the data after they have been transferred on the client side. Concurrency (multithreading) is also implemented to enable the data to be delivered in parallel. Various file sizes were used to test our methodology. This method can be used in different enterprises or governmental institutions to ensure their data are not being manipulated during the transmission process. Furthermore, will also serve as a reference for engineers who will be responsible for implementing integrity measures in their systems.

Keywords: Concurrency; Data integrity; Checksum; Data security; Multithreading; Server-Client file transfer.

1. INTRODUCTION

Many of the distributed applications in public and non-public institutions have faced with the problems of the transfer file functions [1-3]. Nowadays, the biggest problems with data is their integrity during their transmission by using large files. A seriously situation comes for medical records of patients where it should pay attention from the data change during transmission. This would lead to a catastrophe for the patients. Furthermore, this problem is also vital for all businesses and government institutions that their data were not manipulated. This has led to increased attention to data integrity.

There are a lot of studies and research works done in this field but as the technology for ensuring data integrity advances, the same happens with the technology that attackers use to crack data integrity. Some of the methods have been used for transferring large files between a client and a server for ensuring data integrity [4-6]. Based on it, many questions need to be answered related to data integrity for implementing new effective approach.

In this paper we will be focused to provide a method for ensuring the integrity of the data throughout their transfer from the client to the server by computing the checksum of the data on the server and comparing it with the checksum of the data after they have

been transferred on the client side. Concurrency (multithreading) is also implemented to enable the data to be delivered in parallel. Various file sizes were used to test our methodology.

2. THEORETICAL BACKGROUND

Data integrity focuses on data in other tables that are in harmony with each other while performing operations such as updating, deleting, or adding data in a table, thus ensuring that data consistency is not lost [7]. Ensuring data integrity can be done in two ways:

- Identifiable data integrity: the data integrity can be provided by the properties of the defined objects.
- Procedural (Programmatic) data integrity: Integrity must be designed with a programming logic. In SQL, this approach is done with Trigger (triggers), Stored Procedure (stored procedures) or programmer codes.

2.1 Identifiable Data Integrity

The data integrity obtained due to the definitions of the objects is called definable data integrity. There are three types of Constraints (restrictors), Rules (rules) and Defaults (defaults). Identifiable data integrity is more useful and auditable than procedural data integrity; however, in cases where identifiable data integrity cannot be used, procedural data integrity is used.

2.1.1 Identifiable Data Integrity Principles

We can collect definable data integrity principles under three headings.

- Line Integrity (Entity). It is the integrity that allows each record to be a different value from the records entered in the table.
- Column Integrity (Domain). It is the integrity that allows any column in the table to get data from which group or whether it can get a NULL value.
- Application Integrity. Suppose that there are two related tables with a primary key and a foreign key. It is the integrity that ensures that records with the same primary key cannot be deleted from the table with the primary key, in case a record cannot be deleted from the table with foreign key.

There are two basic ways to ensure data integrity in DBMS; Identifiable data integrity and procedural data integrity. Identifiable data integrity is the data integrity that can be provided by the properties of the defined objects. It does not provide much intervention and programming, but it is a very simple definition to use. Procedural data integrity requires designing the integrity with a programming approach. Procedural items are Triggers, Stored Procedures or high-level programs written by programmers [8].

2.2 Checksum

Checksum is the result data obtained by running the Cryptographic hash function algorithm. If you download a file from a source and get the same result when you compare the Checksum provided by the source with the result of the Checksum you

have, it can confirm that the downloaded file and the file in the source are exactly the same and downloaded without error. Checksum can be called hash sum, hash value, hash code, or hash for short, by some sources [9].

Checksum works as a control mechanism that can be useful for computer users in many situations. In case that you downloaded a large file, it can be an ISO installation file of a Linux OS distribution, a Service Pack, or any other file. Based on such large files, you will see a Checksum code on the source page where you downloaded the file. Afterward you can download the file to your computer and getting a value from the downloaded file with the Checksum calculation method [10]. If the Checksum value of the downloaded file matches the one on the download page, it means that you downloaded the file without error.

Furthermore, in case that you are downloading an ISO file (or any file) from a different source other than official sources, you can check whether the file you download is exactly the same as the official source, thanks to Checksum. Due to it you can keep yourself safe from malicious code injected or modified files.

2.3 Concurrency (Multithreading)

Multi-Threaded Programming is the ability to do more than one job at the same time in a program. In other words, while a piece of code is performing an operation, it means that another piece of code works in parallel with it at the same time. A channel (thread) for each employee working in parallel.

Java basically supports multi-channel programming. Multichannel has not been added to the language with some libraries. Every Java program runs in one channel (thread). For example, by running the main method of applications, a main thread named 'main' is run. However, when there is only one channel, the programmer does not realize that the code he writes is running in a thread. Thread provides us the opportunity to do more than one job simultaneously, namely multitasking [11].

Thread benefits are:

- Resource sharing
- Less waiting time
- More efficient hardware usage (Multiprocessor usage)

When a new thread is created, the thread does not start working unless we give a start command, it is ready. It starts running when we give the command to start. In addition, we can hold the thread for a certain time with the sleep command and kill it with the stop command. But what is the difference between thread and multithread?

As we can depict in Figure 1, although threads use the same code, data, and files, they have different registers and stacks. In this way, it can do different jobs within the same code and keep different things in memory. There are two ways to create threads in Java. It has been used it by implementing it. However, threads can be created by extending. There is also a concept called thread pool. You are giving a certain number of threads. Thread pool uses idle thread. One of the most important problems with multithreading is concurrent and synchronization problem. You have to add and delete at the same time. Otherwise, the question arises as to which element you will get without adding the one you want. There are 3 ways to solve this:

- Producer-consumer problem: We can solve it with the same logic that we learned in the operating systems lesson with the producer-consumer problem.

- Circular buffer: Producer-consumer problem is a similar solution. The difference is that you create an array yourself and the lock control happens in this array.
- Java collections: We can leave this to java without dealing with locks ourselves. We can use the collections class that Java provides its own solution to these issues.

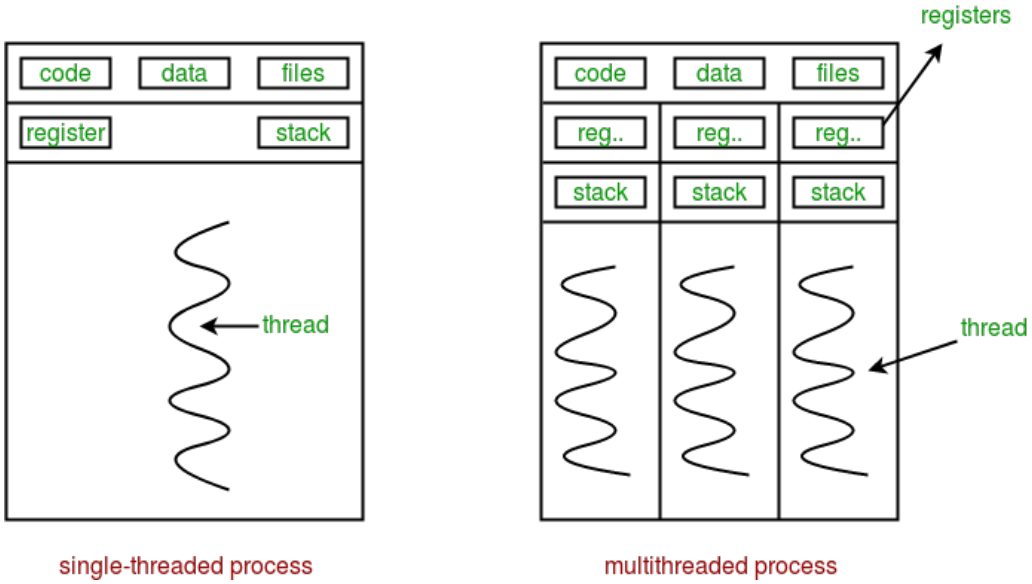

Figure 1. Single Thread vs Multithread [12]

3. OUR APPROACH

In this section we have implemented a system in JAVA in order to make the experiments and testing. We have created a server and a client and by using sockets we will transfer data between them. In our approach, the client will be the one that will define the number of threads that wants to be used for file transfer from client to server. This will indicate how many files will be send from client to server at the same time, of course if the number of files that will be sent is enough. In order to check this and to test the end of the file, we use the following formula:

$$pageNumber*concurrency + listOfFiles.length\%concurrency \quad (1)$$

Together with the file the client sends to server also the file name where at the server side we will have an identical copy of the file. Also, together with the file is sent its checksum at the server-side, when the checksum is calculated, it will be compared with the checksum sent from client side and see if they are the same. If they are the same, it means that the information into the files have not been corrupted, otherwise, if the checksums value is not the same the server will request from the client to send again the specific file.

3.1 Calculation of the Checksum

The file is sent as a metadata together with the calculated checksum from Client to Server. First, we put all the files that we want to transfer in a Queue and we create a thread for each file at the time that they are transferred.

At server side we include the received files in a buffer and then calculate the file checksum and compare if this checksum is the same with the checksum that came with the file from client. We have serialized file metadata and also have allocated 4 bytes for megabytes in the buffer. We have created an HexaArray in order to convert the bytes into a hexa value for the checksum.

3.2 Disadvantages

Related to our approach, there is also a disadvantage which is related to timing. When it comes to large files that are needed to be transferred from server side to client side, the calculation of the checksum for each thread that the client will define to provide concurrency takes a lot of time. This directly indicates the performance of the algorithm, even we have reviewed the complexity of it several times. In order to reduce the time that is needed to calculate the checksum for each file, we are considering of implementing the approach in a different technology (Python), since Java is a slow technology.

4. EXPERIMENTAL SETUP

To test the proposed approach, we have performed the implementation with 3 different scenarios.

I. A dataset of 100 files each with 10MB: These files were sent first without calculating the checksum to the server and then together with the calculated checksum. Also, this scenario was implemented with different concurrency in order to see the indication of the checksum in time.

II. A dataset of 10 files each with 1GB: These files were sent first without calculating the checksum to the server and then together with the calculated checksum. This second scenario was implemented with different concurrency in order to evaluate the impact of the checksum in time.

III. The third scenario considers a mixed dataset of 10 files with 1GB and 100 files each with 10MB: As it can be seen, this is a combination of the two previous scenarios. The reason why we run this scenario is to test our approach with different file size. These files were sent first without calculating the checksum to the server and then together with the calculated checksum. Even in this case the concurrency varies and the checksum versus time is evaluated.

By running these experiments, we tested the time needed to transfer the files, the accuracy of our checksum approach and also the indication that the checksum calculation has in the time needed to transfer the files.

Testing files were randomly created by using Python programming language. The data stored in these files are not real-world data, but they are used just for testing. The client communicates with the server by using a specific port (in our case 9999) in the socket that we have created.

In the approach that we have presented, among the file transfer by using multi-threading techniques, that is widely used for data and file transfer, we have calculated the checksum of each file for each thread, to ensure the integrity of the file when it reaches the client side. Besides this, when the algorithms checks if the checksum of the files is the same at client side, it also compares the file name and if one of these to comparisons has a negative result, it requests from the server to resend the file. Until now, these techniques were not seen during the literature review. The approach can be improved, so in case that some file is lost or damaged in the server side, by keeping

track on the clients that the file was sent, the server can request the file so it can restore it.

5. EXPERIMENTAL RESULTS

In this section are analysed the results for all the scenarios described in the previous section.

5.1 Scenario with 100 Files x 10MB Dataset

Figure 2 depict concurrency that is equal to 1 where throughput is 23 s. When concurrency is 2 the throughput corresponds to 21 s. If concurrency is equal to 4 the throughput is 19 s and, when concurrency is 8 the throughput is 17 s. As we can see from the results, when the concurrency (number of threads) increases, the time needed to transfer these files decreases and in all cases the accuracy is 100%, which means that the files in client-side and server-side are completely the same.

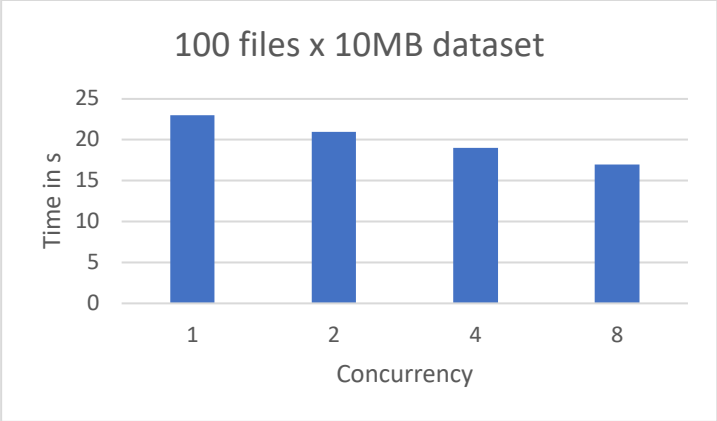

Figure 2. Results for the scenario with 100 Files x 10MB Dataset.

5.2 Scenario with 10 Files x 1GB Dataset

In Figure 3 are shown concurrency which is equal to 1 and the throughput is 196 s, when concurrency is 2 the throughput is 188 s. If concurrency is equal to 4 the throughput is 183 s, and when concurrency is 8 the throughput is 176 s. As the number of threads increases, the time again decreases. Also, the accuracy of the checksum is 100%, which means that the files were the same in client and server side.

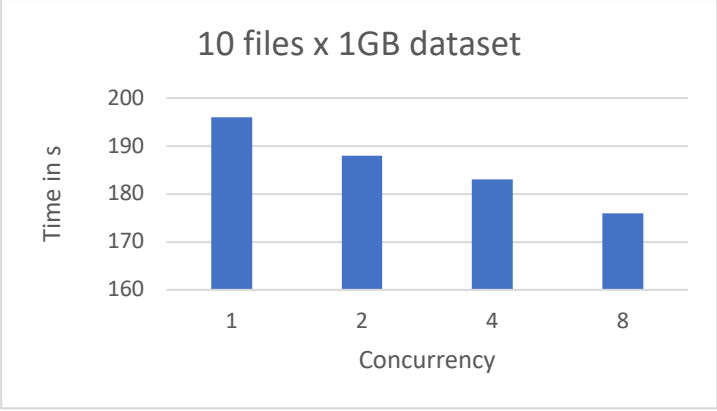

Figure 3. Results for the scenario with 10 files x 1GB Dataset.

5.3 Scenario with 10 Files x 1GB + 100 Files x 10MB Dataset

In the Figure 4 it has been shown that concurrency is equal to 1 and the throughput is 1220 s. When concurrency is 2 the throughput is 1130 s. If concurrency is 4 the throughput is 1072 s. And last when concurrency is equal to 8 the throughput is 1037 s. As we can see, as the number of threads increases, the time needed to transfer the files decreases and in terms of checksum, the accuracy is 100 % in all the cases. The files are identical in both server and client side.

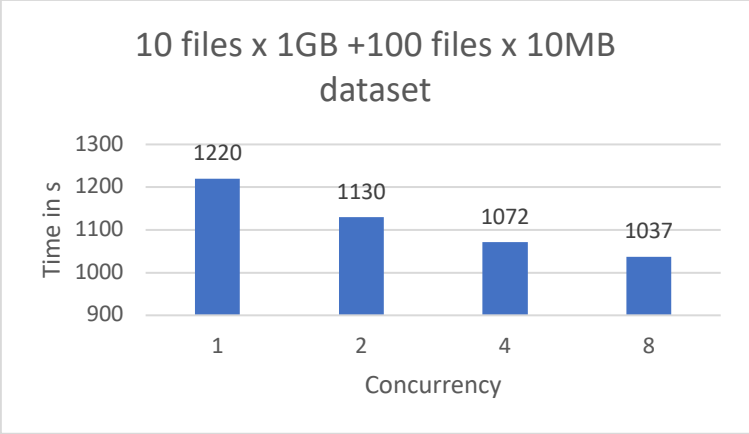

Figure 4. Results for the scenario with 10 files x 1GB + 100 Files x 10MB Dataset.

6. CONCLUSION

In this paper, we have provided a method for ensuring the integrity of the data throughout their transfer from the client to the server by computing the checksum of the data on the server and comparing it with the checksum of the data after they have been transferred on the client side. In our experimental results of the tests that we made with different datasets, due to the concurrency number is increased it has been seen a tendency of time decrease for sending the files from Client side to Server Side. In all scenarios the accuracy of the checksum is 100 % due to the increasement of the concurrency number. The files were in all cases the same in both server and client side. There is a small increase in the size of the files sent from client to server due to the send of the checksum calculation together with the file. It is obvious that since the size of the file is increased, we have seen that the time is also increased.

In this approach we have seen that checksum calculation play an important role for assuring the information integrity. Our research work needs to be tested in real world for assuring the accuracy of the implementation of this approach in real world would be the same.

CONFLICT OF INTERESTS

The authors would like to confirm that there is no conflict of interests associated with this publication and there is no financial fund for this work that can affect the research outcomes.

REFERENCES

[1] Forouzan A.B. (2003) Data communication and networking," Tata McGraw-Hill Publication, 2nd ed., USA

[2] Bender W. Techniques for Data Hiding," *IBM Systems Journal*, 1996; 35(7); 313-336.

[3] Joel L.O., Doorsamy W. & Paul B.S. A Review of Missing Data Handling Techniques for Machine Learning. *International Journal of Innovative Technology and Interdisciplinary Sciences*, 2022; 5(3); 971–1005.

[4] Kumar V.V. and Poornima G. Ensuring Data Integrity in Cloud Computing, *Journal of Computer Applications*, 2012; 5(4); 513-520.

[5] Zarour M., Alenezi M., Ansari T.J., Pandey A.K., Ahmad M., Agrawal A., Kumar R. and Khan R.A. Ensuring data integrity of healthcare information in the era of digital health. *Healthcare Technology Letters*, 2021; 8; 66-77.

[6] Zhang P., White J., Schmidt D.C, Lenz G. and Rosenbloom S.T. FHIRChain: Applying Blockchain to Securely and Scalably Share Clinical Data. *Computational and Structural Biotechnology Journal*, 2018; 16; 267-278.

[7] Lagoze C. Big Data, data integrity, and the fracturing of the control zone, *Big Data & Society,* 2014; 1(2); 1-11.

[8] Nikam N.R., Patil P.R., Vakhariya R.R., Mohite S.K. and Magdum C.S. Data Integrity: Overview, *International Journal of Recent Scientific Research,* 2020; 11(6), 38762-38767.

[9] Trahay F. Brunet E. and Denis A. An analysis of the impact of multi-threading on communication performance, *IEEE International Parallel & Distributed Processing Symposium*, 2009, pp. 1-7.

[10] Smoak C.G., Widel M. and Truong S. The use of checksums to ensure data integrity in the healthcare industry, *Pharmaceutical Programming*, 2012; 5(1/2); 38-41.

[11] Smoak C.G. Checksum Please: A Way to Ensure Data Integrity, *PharmaSUG*, 2012, pp. 1-4.

[12] Multithreading in Python. Available at https://www.geeksforgeeks.org/multithreading-python-set-1/. Accessed on 15 June 2022