Shadow Data: A Method to Optimize Incremental Synchronization in Data Center

Changjian Zhang, Deyu Qi, Wenhao Huang

Shadow Data: a Method to Optimize Incremental Synchronization in Data Center

Changjian Zhang[1], Deyu Qi[1✉], and Wenhao Huang[1,2]

[1] School of Computer Science and Engineering, South China University of Technology
csa@scut.edu.cn
[2] Guangzhou Mingsen Technology Company Ltd., Guangzhou 510000, China

Abstract. With the continuous increase of data, the data center that plays the role of backup is facing the problem of energy hunger. In practice, to reduce the bandwidth, the local data is synchronized to the data center based on incremental synchronization. In this process, the data center will generate a huge CPU load. To solve this pressure of the data center, first, we analyze the process of the Rsync algorithm, the most commonly used in incremental synchronization, and CDC algorithms, another way of chunking algorithm. Then we propose a data structure called Shadow Data, which greatly reduces the CPU load of the data center by sacrificing part of the hard disk space in the local node.

Keywords: Data Synchronization·Shadow Data·data backup

1 Introduction

In case of the loss of important data caused by the PC crash, companies and individuals choose to put these data in their own or third-party data centers[1, 2]. However, data centers face challenges from data synchronization and others[3, 4]. As more and more data is managed in data centers, more and more requests for data synchronization will be made. In this case, data centers usually increase their processing capacity by adding servers.

Since most of the synchronization requests in the data center are incremental synchronization, to increase the parallel processing capacity of the data center, we start with the process of incremental synchronization. We analyze why incremental synchronization would cost a lot of data center resources, and remove the most CPU consuming steps from the process, store the Shadow Data of backup in the local node instead.

The main contributions of this paper are as follows:

1. We analyze the process of incremental synchronization algorithms, including the Rsync algorithm and CDC(content-defined chunking) algorithms. Then we find out the steps caused CPU load of the data center in this process.
2. We propose a data structure, Shadow Data. After finding out the cause of the CPU load in the data center, we propose to store the Shadow Data of the data center in the local hard disk, which can cut off several steps of incremental synchronization at the data center to reduce the CPU load.

3. In the experiments, we test the practicability of Shadow Data. For the same size of local data that you want to synchronize to the data center, the Shadow Data can reduce the CPU load of the data center to about 14% of the original.

2 Background and Related Work

2.1 Incremental Synchronization

With the growth of the size of one single file, the synchronization of the file always goes incremental. When incremental synchronization is implemented, it is achieved through multiple network communications between the local and data center. The simplified communication flow of incremental synchronization between local and data center is shown in Fig. 1[5, 6]. It should be noted that the synchronization process discussed in this paper is the upload process of one single file in the data center. The upload process refers to copying the local data to the data center. Corresponding to this is the download process, which copies the data in the data center to the local.

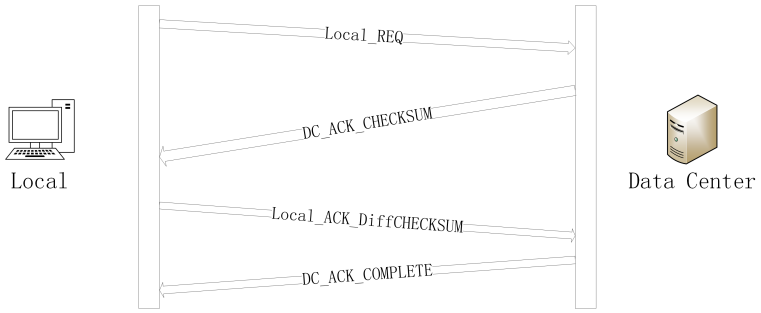


Fig. 1: The communication flow between the local and data center to complete the synchronization of one single file.

As mentioned above, the communication process shown in Fig. 1 is simplified. We remove some confirmation messages and focus on the key messages the four in Fig. 1. A detailed introduction to these messages is as follows:

1. Local_REQ: Request message. It is sent to the data center from the local node, indicating that a synchronization request about one single file is to be initiated. The message content is: $|LocalFileInfo|DCFileInfo|$. $LocalFileInfo$: information about local file in this synchronization request. $DCFileInfo$: information about target file in the data center.
2. DC_ACK_CHECKSUM: message sent by data center to local. It contains chunk abstract information for the target file stored in the data center. The message content is: $|Over\_flag|Chunk\_abstract|$. $Over\_flag$ indicates whether it is the last message. $Chunk\_abstract$: abstract information for one chunk.

3. Local_ACK_DiffCHECKSUM: message sent by local to data center. After comparing the local file and the checksum content, local node gets the different chunk information, and sent to data center with $Local\_ACK\_DiffCHECKSUM$ message. The message content is: $|Over\_flag|Chunk\_info|$. $Over\_flag$ indicates whether it is the last message. $Chunk\_info$: chunk information. There are two kinds of chunk information: one for the same chunk and the other for the different chunk.
4. DC_ACK_COMPLETE: message sent by data center to local. This message is to tell local that the synchronization request is finished. The message content is: $|Complete\_info|$. It contains some information about result.

The actual message sent is $messagehead + messagecontent$. $messagehead$: $|Mes_head|Mes_type|$. Mes_head contains some version information. Mes_type is the type of this message.

2.2 Related Work

Academic circles have done a lot of research on incremental synchronization among files. The Rsync algorithm[5], proposed by Andrew Tridgell, claims to complete the synchronization task through multi-segment communication and propose a strong and weak hash code to improve synchronization performance. The weak hash function chosen by Rsync is the Adler-32, which is a rolling-check algorithm, and the strong one is MD5. To optimize the resource usage of Rsync, Chao Yang et al. proposed an optimized communication process to reduce the data center CPU load during downloading[6]. Besides, many scholars optimize the Rsync algorithm from the perspective of the chunking algorithm. Won Youjip et al. proposed MUCH algorithm base on Rabin to speed up the chunking process with multi-thread[7]. Jihong Ma et al. proposed UCDC algorithm, claims the definition of a chunking mark is: for a value of a string, taking the remainder from being divided by a fixed value[8]. Instead of division, some chunking algorithms using byte comparison are proposed: LMC(Local Maximum Chunking) algorithm decides to set a cut-off point when the maximum value of a window data is in the middle of the window[9]; in order to speed up the validation of the window data, AE[10] and RAM[11] algorithms are proposed. For AE, if the maximum value of bytes in the data window is located at the end of the window, the cut-off point is set at the end of the window. For RAM: if a byte value with no less than all byte values in the window is read out of the window, a cut-off point is set at this byte. To make the cut point of chunk more stable, Changjian Zhang et al. proposed MII[12] and PCI[13] algorithms, set a cut-point based on the length of a increasing interval and number of '1' in binary window Separately.

However, the focus of these studies is to improve the synchronization performance without considering the optimization of synchronization process and reducing unnecessary steps with the idea of space for time.

3 The Design of Shadow Data

3.1 What Makes the CPU Load in the Data Center

Before explaining the design of the Shadow Data, first, we discuss the reasons for the CPU load in data centers. As shown in Fig. 1, $DC\_ACK\_CHECKSUM$ messages are sent by the data center. Before sending, the data center needs to chunk the backup files and calculate the summary information. These processes generate a lot of calculations, especially in the calculation of strong checksum, such as MD5.

3.2 Why Shadow Data

For the backup system, when a local file is synchronized to the data center, at this point, the local and data center are consistent. At the next time point, the local data has changed and needs to do one synchronization. However, the local does not need to ask the data center for the content of the local data at the previous time point, because the content once existed at its own disk.

For example, as shown in Fig. 2, the content of the backup file in the data center at t1 time is the content of the local file at t0 time, and then after synchronization, the content of the backup file becomes the content of the local at t1 time. At t2 time, if the local file wants to synchronize for one time, it needs to know its data at t1 time. In the normal communication process, it needs the $DC\_ACK\_CHECKSUM$ messages from the data center. As shown in Fig. 2, we can see that the data of local at t1 time can also be obtained from its previous records.

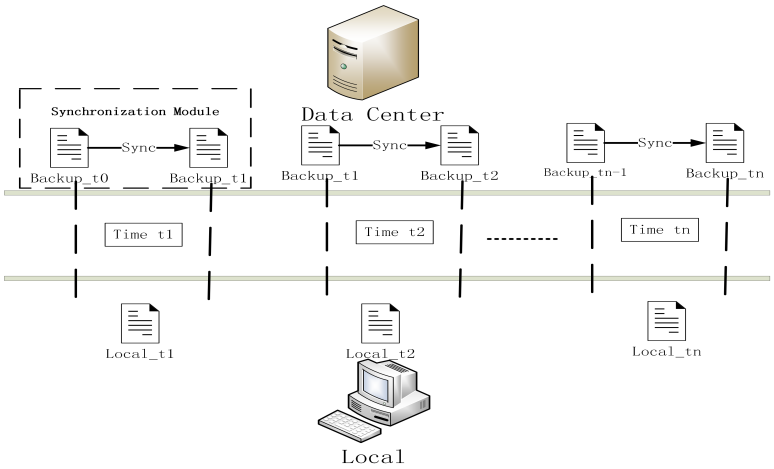


Fig. 2: The change of files in both local and the data center during synchronization of one single file.

3.3 What is Shadow Data

The comparison of the "synchronization module" shown in Fig. 2 before and after using Shadow Data is shown in Fig. 3. $AbstractInfo$ denotes $DC\_ACK\_CHECKSUM$ message, and $DiffInfo$ denotes $Local\_ACK\_DiffCHECKSUM$ message.

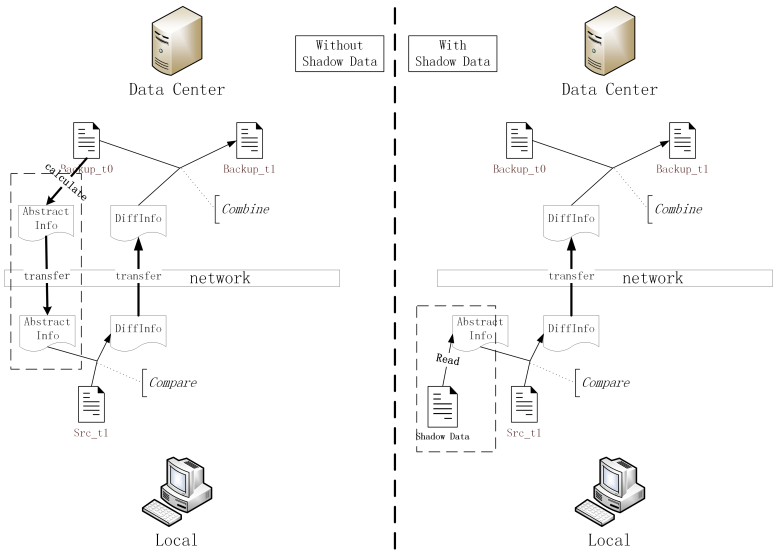


Fig. 3: The comparison of the "synchronization module" with and without using Shadow Data.

Shadow Data is proposed to replace the abstract information of backup files sent from the data center. For example, at t1 time, when the local node completes a synchronization, it stores its own summary information on its disk. Thus, when it needs to synchronize at t2 time, it will no longer need to ask data from the data center, which removes the most CPU load of the data center and, at the same time, saves one data transmission. In practice, the amount of data transmitted this time is about half of the total amount of data transmitted during synchronization. The abstract information stored on the local disk here is called Shadow Data, the shadow of the backup file.

The Shadow Data format is a chunk abstract information list, which is stored in the order of the backup file chunks. The specific format is $AbstractInfo$; $AbstractInfo$; ...$AbstractInfo$;. Where $AbstractInfo$ is in the form of: $ChunkAbstract, StartIndex, ChunkLength$. $ChunkAbstract$ refers to the strong checksum of chunks, such as MD5, and the weak check if needed; $StartIndex$ indicates the starting index of the chunk in the backup file; $ChunkLength$ indicates the length of the chunk. Based on this design, in the $Local\_ACK\_DiffCHECKSUM$ message, the format of $Chunk\_info$ is: $1, ChunkIndex, BlockLength$, if it is the

same block, and $0, ChunkData$, if it is a different one. In the data center, after receiving $Local\_ACK\_DiffCHECKSUM$ message, the new backup file can be merged only by random reading without calculation.

The storage format of the Shadow Data is shown in Fig. 4, and the format of $Local\_ACK\_DiffCHECKSUM$ message is shown in Fig. 5.

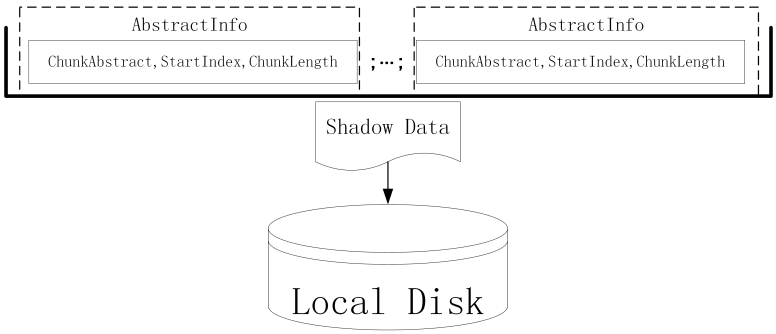


Fig. 4: Storage format of Shadow Data.

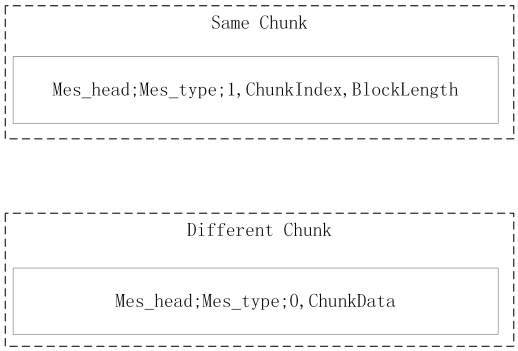


Fig. 5: Format of $Local\_ACK\_DiffCHECKSUM$ message.

It is worth noting that Shadow Data is only applicable to data centers that play a backup role, not data centers that play a sharing role. Because the latter will have multiple different clients to update the data in the data center.

4 Evaluation

4.1 Experiment Setup

The experimental environment of the data center is shown in Table 1. For the local environment, we only implement a client on another machine, connecting the port of the data center for interaction.

Table 1: Experimental environment

Operating System	CPU	Memory	IDE
Windows 10	i5-7500 3.40GHz	16g	Idea2019.3

The datasets are randomly generated data. Three files with a size of about 2.1G are generated by the Mersenne Twister Pseudo-Random Number Generator[14]. The three files are divided into three groups. These three files act as backup files. In each group, the backup file is randomly modified to generate a new file, which will act as the local file. The experimental datasets are shown in Table 2.

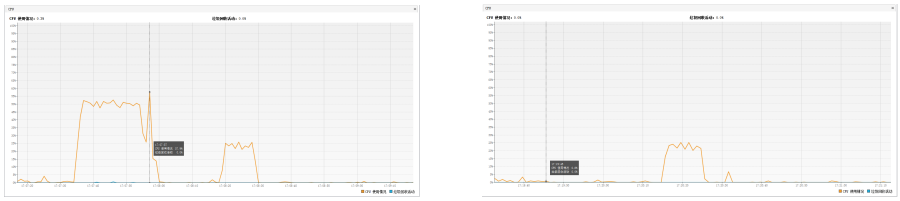
Table 2: The datasets

Group Id	Backup file	Local File
Group1	2.1G	about 2.1G
Group2	2.1G	about 2.1G
Group3	2.1G	about 2.1G

In the experiment, we use the CDC algorithms instead of Rsync, and the type of CDC algorithms is PCI algorithm. Since we only simulate the incremental synchronization process, using Shadow Data to reduce the steps of the data center, the choice of synchronization algorithm will not affect the final experimental results and the same to the datasets.

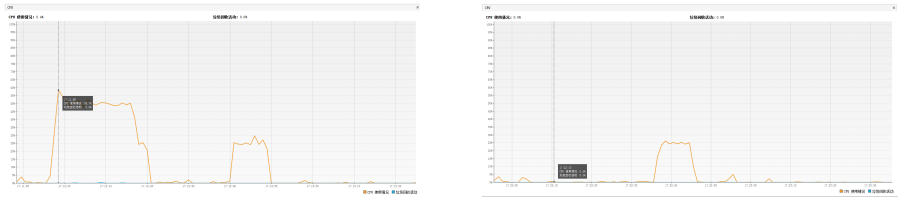
4.2 CPU Load of the Data Center

We have carried out the synchronization process for three groups of data respectively, and then monitored the CPU utilization of the thread responsible for synchronization in the data center with the help of the monitoring tool. The



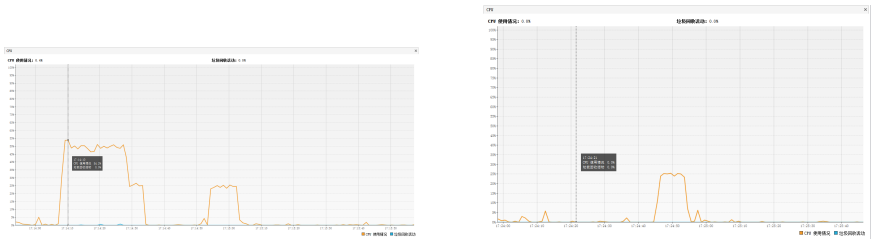
(a) (b)

Fig. 6: CPU load in the data center without((a)) and with((b)) Shadow Data based on the group1 dataset.



(a) (b)

Fig. 7: CPU load in the data center without((a)) and with((b)) Shadow Data based on the group2 dataset.



(a) (b)

Fig. 8: CPU load in the data center without((a)) and with((b)) Shadow Data based on the group3 dataset.

experimental results of the three groups of data are shown in Figure 6, 7 and 8 separately.

Let's focus on one of the three groups of test data, figure (a) shows the situation when there is no Shadow Data. In this case, the data center needs to calculate the abstract information of the backup file, and the resulting CPU utilization is shown as the first irregular polygon in the figure. The second irregular polygon in the figure means the CPU utilization for generating new files, the

same as the first one in figure (b). From the two irregular areas, we can see that the CPU utilization generated by calculating the abstract information of the backup file is much higher than the CPU utilization generated by generating the new file, which is more than six times. In other words, with Shadow Data, the CPU utilization of the data center with single synchronization can be reduced to about 14% of the original. The reason is obvious, since there is no need to calculate the abstract information of the backup file with Shadow Data. Since the actual CPU load is not easy to capture in the running of the algorithm, so in this paper, the CPU utilization is used to approximate the CPU load, which may not be rigorous, but in qualitative analysis, it makes sense.

4.3 Sacrifice of Disk on Local Node

The use of Shadow Data makes it necessary to store part of the data locally, which is put on the local disk. When the local wants to initiate one synchronization, it will be read from the local. In the experiment, we use memory mapping to read, which speeds up the reading speed. In Table 3, we show the cost of local disks corresponding to three groups of test data.

Table 3: Cost of Local Disk

Group Id	number of chunks(piece)	cost of disk(KB)
Group1	2508184	68583.2
Group2	2391516	65393.0
Group3	2446812	66905.0

The size of disk usage is related to the number of chunks. In this experiment, the average block size[9] is used, which is close to the average block size of Rsync. It can be seen that for about 2.1G backup files, about 65M local disk capacity needs to be consumed to store Shadow Data, aka the abstract information. However, this saves a transmission of this data from the data center. In fact, if it is sent from the data center, the data sent should be greater than 65M, because the message header should also be included. In the multi round communication proposed by Rsync algorithm, the client and the server must be dual channels, which can not be guaranteed in some applications. Using shadow data can simplify the communication process of file incremental synchronization, which only needs a single channel from client to server to complete one synchronization. The results of synchronization can be obtained by querying the server by the client.

5 Conclusion

By analyzing the process of incremental synchronization algorithm, we find out the steps, which generate the server-side CPU load. Shadow Data is proposed to remove these steps. Shadow Data is stored on the disk of the local node to replace the abstract information of backup files, which should be sent from the data center. In the experimental part, we verify the practicability of Shadow Data.

References

1. Elahi, B., Malik, A.W., Rahman, A.U., Khan, M.A.: Toward scalable cloud data center simulation using high-level architecture. Software: Practice and Experience **50**(6) (2020)
2. Tang, X., Wang, F., Tong, L.I., Zhang, P.: Research and implementation of real-time exchange system in data center. Computer Science (2017)
3. Nizam, K.K., Sanja, S., Tapio, N., Nurminen, J.K., Sebastian, V.A., Olli-Pekka, L.: Analyzing the power consumption behavior of a large scale data center. Computer Science Research & Development (2018)
4. Zhi, C., Huang, G.: Saving energy in data center networks with traffic-aware virtual machine placement. Information Technology Journal **12**(19) (2013) 5064–5069
5. Tridgell, A.: Effcient algorithms for sorting and synchronization. `https://www.samba.org//tridge/phd\_thesis.pdf` Accessed February, 1999.
6. Chao, Y., Ye, T., Di, M., Shen, S., Wei, M.: A server friendly file synchronization mechanism for cloud storage. In: IEEE International Conference on Green Computing & Communications, IEEE & Internet of Things. (2013)
7. Won, Y., Lim, K., Min, J.: Much: Multithreaded content-based file chunking. IEEE Transactions on Computers **64**(5) (May 2015) 1375–1388
8. Ma, J., Bi, C., Bai, Y., Zhang, L.: Ucdc: Unlimited content-defined chunking, a file-differing method apply to file-synchronization among multiple hosts. In: 2016 12th International Conference on Semantics, Knowledge and Grids (SKG). (Aug 2016) 76–82
9. Bjørner, N., Blass, A., Gurevich, Y.: Content-dependent chunking for differential compression, the local maximum approach. J. Comput. Syst. Sci. **76**(3-4) (May 2010) 154–203
10. Zhang, Y., Feng, D., Jiang, H., Xia, W., Fu, M., Huang, F., Zhou, Y.: A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems. IEEE Transactions on Computers **66**(2) (Feb 2017) 199–211
11. Widodo, R.N.S., Lim, H., Atiquzzaman, M.: A new content-defined chunking algorithm for data deduplication in cloud storage. Future Generation Computer Systems **71** (2017) 145–156
12. Zhang, C., Qi, D., Cai, Z., Huang, W., Wang, X., Li, W., Guo, J.: Mii: A novel content defined chunking algorithm for finding incremental data in data synchronization. IEEE Access **7** (2019) 86932–86945
13. Zhang, C., Qi, D., Li, W., Guo, J.: Function of content defined chunking algorithms in incremental synchronization. IEEE Access **8** (2020) 5316–5330
14. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. Acm Transactions on Modeling & Computer Simulation **8**(1) (1998) 3–30