2012 International Conference on Image, Vision and Computing (ICIVC 2012)
IPCSIT vol. 50 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V50.19

A Reverse Differential Archiving Method based on Zdelta

CHEN Gang[a,*]

[a] Dept. of Computer Science Sichuan University, Chengdu and 610065, P.R. China

Abstract. In order to improve the recovery efficiency of archives and reduce the space occupied by the backup files, a zdelta compression algorithm based method of reverse differential archive is proposed. Aiming at people's habits of recovering files--usually recovering the files with recent point in time, this archiving task is achieved by calculating the forward and reverse difference set and reconstructing the latest files with differential files by improving zdelta compression algorithm. The experiment results indicate that this method can decrease recovery time and space needed remarkably, and improve efficiency of file recovery evidently over traditional forward file archive.

Keywords: file archive, reverse difference calculation, file reconstruction, file recovery, zdelta

1. Introduction

Rsync[1, 2] is a remote data synchronization tool of Unix-like operating system. It uses "Rsync algorithm" to enable local and remote files to achieve rapid synchronization. In reducing the server's storage space, zdelta[3] mentioned in Dimitre Trendafilov, Nasir Memon and Torsten Suel's paper, main idea just as in vdelta[4]/vcdiff[5], is to represent the target data as a combination of copies from the reference data and the already compressed target data.

However, these traditional differential archiving method making file recovery will perform a lot of refactoring operations and cost a lot of time and server resources. Because recovering the archive file, people will often choose to restore the last time archive file. Thus, it needs to take multiple refactoring operations to reconstruct the needed archive file with the first full-backup file and each difference files. The time cost will rise with the increase of the number of reconstruction and this may cause great consumption of resources. Thus, there is a need to improve the current archive management techniques to make the backup file fast recover and reduce the space occupied.

This paper will simplify and improve zdelta to generate the forward difference and reverse difference between the files at the same time. Then we can use the forward difference file and the last time archive file to reconstruct the latest file mirror, and use the reverse difference file for archive management.

2. The Fundamentals of Reverse Differential Algorithm

2.1. Reverse differential algorithm analysis

2.1.1. Introduction to reverse different archiving method

The traditional differences archiving method to store the archive file is shown as Figure 1. They only save the full backup file of time T_0 and forward differences from time point T_1 to T_{n-1}. Using these methods to recover the recent time point file requires multiple reconstruction operation. For example, in order to recover the latest archive file of time T_{n-1} requires n-1 times of reconstruction; To recover the second latest file of time T_{n-2} needs n-2 times of reconstruction.

* Corresponding author.
E-mail address: shouhuxing1@foxmail.com.

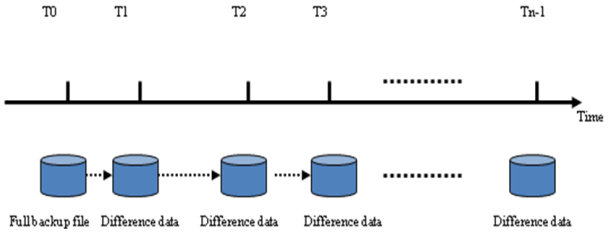

Fig. 1. Diagram of the traditional archiving storage

To recover the archive files, people often choose to recover the files with recent point in time. Thus, traditional archiving methods need to take multiple refactoring operations to reconstruct the needed archive files. So we generate both the forward and the reverse difference between the adjacent files through only once scan and comparison at the same time on the basis of simplifying the zdelta algorithm. Use the forward difference to reconstruct a copy of the newest file in the archiving server and save the copy. When recovering archive files, you can use the reverse difference to reconstruct the file needed. This way, to recover the latest point archive file, directly transmit the latest copy on the server to the client without the need for difference reconstruction; To recover the second latest file needs only once reconstruction. The reverse differential archiving method to store the file is shown as Figure 2. The figure shows, this reverse differential archiving method saves the forward and reverse difference files from time T_0 to T_{n-2} and the full backup file of time T_{n-1}. If you choose to recover the newer archive files, the reverse differential method can significantly reduce the count of archive files reconstruction.

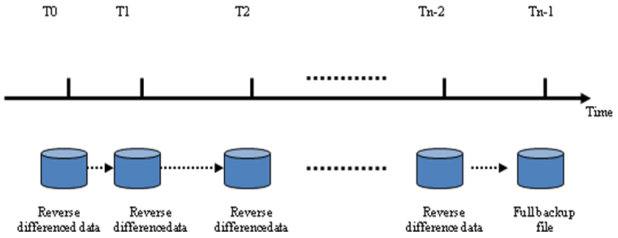

Fig. 2. Diagram of the reverse differential archiving storage

2.1.2. Definition of terms[6]

File f_i, representing user's archiving file.

- Archive file set $F = \{f_1, f_2 \ldots f_i \ldots f_n\}$, multiple archive files or directories that the composition of the collection.
- Forward differences set $\Delta F = F_2 - F_1$, recording the differences between the archive file set of time t_1 and t_2, $t_1 < t_2$.
- Forward difference file pair $<f_p, f_d>$, recording the differences of files between the adjacent archive point, $<f_p, f_d> \in \Delta F$. f_p is the difference instruction file, recording the length of differences data and the location flag of the match string, differences data length stored as a positive integer while matching string flag stored as a negative integer -1 in the reference file and -2 in the compressed target file. f_d is the forward difference data file, recording the difference data between the reference file and the target file, and the offset and length of match string, difference data and match length stored as a byte, the offset of match string stored as two bytes.
- Reverse differences set $\Delta F' = F_1 - F_2$, recording the differences between the archive file set of time t_2 and t_1, $t_1 < t_2$.
- Reverse difference match file f_m', recording the reverse matching degree of archive file between adjacent archiving points, including the length of the match string found in the reference file f_i, the offset of the match in the reference file f_i and the offset in the target file f_i'.
- Reverse difference file pair $<f_p', f_d'>$, recording the reverse differences of files between the adjacent archive point, $<f_p', f_d'> \in \Delta F'$. f_p' is the reverse difference instruction file, recording the length of reverse differences data and matching flag, reverse differences data length stored as a positive integer while matching flag stored as a negative integer -1. f_d' is the reverse difference data file, recording the reverse difference data between the reference file and the target file, and the offset and length of match string in the target file, reverse difference data and match length stored as a byte, the offset of match string in the

target file stored as two bytes.

- Operation $F_B = O(f_i, f_j)$, calculating the differences between the reference file f_i and f_j the target file, and the reverse difference match file f_m'.
- Operation $F_S = O(f_m')$, generating the reverse difference file pair $<f_p', f_d'>$ according to the reverse difference match file f_m'.
- Operation $f_j = O(f_i, f_p, f_d, f_t)$, reconstructing the target file f_j according to the reference file f_i and difference file pair $<f_p, f_d>$. f_t is a temporary file.

2.2. Steps of the algorithm based on zdelta

The differential reverse archiving algorithm can be divided into the following four steps:

2.2.1. Calculate forward difference and reverse match degree

This step is to complete the operation $F_B = O(f_i, f_j)$. The following is the details:

1. Initialize the reference file hash table:

For i = 0 to len(f_{ref}) – 3,

(a)Compute $h_i = h(f_{ref}[i, i+2])$, the hash value of the first three characters starting from position i in f_{ref}.

(b)Insert a pointer to position i into hash bucket h_i of T_{ref}.

2. Traverse the target file f_i' to generate the forward difference file pair $<f_p, f_d>$ and the reverse match file f_m':

Set j = 0;

While j ≤ len(f_{target}):

(1) Compute $h_j = h(f_{target}[j, j+2])$， the hash value of the first three characters starting from position j in f_{target}.

(2) Search hash bucket h_j in both the reference file hash table T_{ref} and the compressed target file hash table T_{target} to find the match string.

(3) If no hash bucket found, there is no string that matches the current string in the reference file and the compressed target file. Record ftarget [j] into fd, then insert a pointer to position j into hash bucket hj of Ttarget. Set j = j + 1, the length of the forward differences data: diffcount = diffcount + 1;

If a hash bucket equal to h_j is found, there is a possible string that matches the current string in the reference file and the compressed target file("possible" refers to the hash of three different characters may be equal.). So compare the two strings character by character, note the continuous and same characters counts as count, then compare all the counts and get the maximum value maxlength. If maxlength ≥ 3, record the forward differences data counts diffcount into f_p, and put the location flag of the match string into f_p(If the match found is in the reference file, put the length of the match maxlength, the reference file offset and the target file offset j into the reverse difference match file f_m'.), then put the offset and the length of the match maxlength into f_d, hash all substrings of $f_{target}[j, j + maxlength - 1]$ and insert them in T_{target}. Set j = j + maxlength, forward difference data counts diffcount = 0. If maxlength ≤ 3, there is no match found. Record $f_{target}[j]$ into f_d, then insert a pointer to position j into hash bucket h_j of T_{target}. Set j = j + 1, the length of the forward differences data: diffcount = diffcount + 1.

If j > len(f_{target}), the forward difference file pair $<f_p, f_d>$ and the reverse difference match file f_m' have been generated successfully.

2.2.2. Generate the reverse difference

After the forward difference file pair $<f_p, f_d>$ and the reverse difference match file f_m' are uploaded to the archiving server, the server generates the reverse difference file pair $<f_p', f_d'>$ according to f_m'. That is to finish the operation $FS = O(f_m')$. Details show as follows:

Set j = 0;

While j ≤ len(f_{ref}):

(1)Search the reference file offset of a match equal to j in the reverse difference match file f_m'.

(2)If there is no match found, record $f_{ref}[j]$ into f_d', set j = j + 1, the length of the reverse differences data: reversediffcount = reversediffcount + 1;

If a match is found, record the length of the reverse difference data: reversediffcount into f_p', put the flag of the match into f_p' also. Then put offset of the target file: offset and the length of the match: maxlength into f_d', set j = j + maxlength, the reverse difference data counts: reversediffcount = 0.

If j > len(f_{ref}), the reverse difference file pair <f_p', f_d'> has been generated successfully.

2.2.3. Calculate the latest archive file

Use the old file f_i and the forward difference file pair <f_p, f_d> to build the new file f_j. That is to finish the operation $f_j = O(f_i, f_p, f_d, f_t)$. Details show as follows:

(1)Read data from f_p. If it has reached the end of the file, finish the operation of calculating the latest archive file, cover file f_i with file ft to get file f_j. Otherwise, get number x from f_p.

(2)If x > 0, get x bytes from current file pointer in fd, then put them into ft. Go to (1).

(3)If x = -1, get 3 bytes from f_d. The first two bytes is the offset of the match: offset in the old file f_i. The third byte is the match length: matchlength. Move the file pointer of f_i to offset, and get matchlength bytes, put them to temporary file f_t. Go to (1).

(4)If x = -2, get 3 bytes from f_d. The first two bytes is the offset of the match: t_offset in the temporary file f_t. The third byte is the match length: t_matchlength. Move the file pointer of f_t to t_offset, and get t_matchlength bytes, put them to temporary file f_t. Go to (1).

2.2.4. Reverse differential recover

Use the copy of the latest archive file f_j and the reverse difference file pair <f_p', f_d'> to build the previous archive file f_i. That is to finish the operation $f_i = O(f_j, f_p', f_d', f_t)$. Details show as follows:

(1) Read data from f_p'. If it has reached the end of the file, finish the operation of calculating the previous archive file, cover file f_j with file f_t to get file f_i. Otherwise, get number x from f_p'.

(2) If x > 0, get x bytes from current file pointer in f_d', then put them into f_t. Go to (1).

(3) If x = -1, get 3 bytes from f_d. The first two bytes is the offset of the match: n_offset in the new file f_j. The third byte is the match length: n_matchlength. Move the file pointer of f_j to n_offset, and get n_matchlength bytes, put them to temporary file f_t.

The file reconstructed f_i is the previous archive file need to recover.

3. Experiment

In this section, we observed the compression performance and recovery efficiency of the reverse differential archiving algorithm based on zdelta. We evaluated our system using the emacs[3, 4] files less than 64K in the lisp folder of version 20.1, 20.2, 20.3, 20.4, 20.5, 20.6, 20.7, 21.1, 21.2 and 21.3 as archive files. All our experiments were conducted on a 1.79 GHz AMD system with 1 GB of main memory running Microsoft Windows XP Professional SP2.

3.1. Experiment methods

We compared the time cost using the forward differential method with using the reverse differential method to recover the backup files to recent archiving point with the increase of the number of archiving points, and recorded the size of the forward and the reverse difference.

(1) Use the reverse differential client algorithm to compare the data set of version 20.1 to 20.2, and generate the forward difference file pair and the reverse difference match file.

(2) Then use the reverse differential server algorithm to generate the copy of the newest archiving point 20.2 and the corresponding reverse difference file pair, record the size of the forward and the reverse data.

(3) Use the reverse differential client algorithm to compare the data set of version 20.2 to 20.3, and generate the forward difference file pair and the reverse difference match file.

(4) Then use the reverse differential server algorithm to generate the copy of the newest archiving point 20.3 and the corresponding reverse difference file pair, record the size of the forward and the reverse data.

(5) Compare the time cost: record the time cost by using the forward differential method to recover the second latest archiving point files 20.2 with old files 20.1 and the corresponding forward difference file pair; and record the time cost by using the reverse differential method to recover the second latest archiving point files 20.2 with the newest archiving point files 20.3 and the corresponding reverse difference file pair.

(6) Use the reverse differential client algorithm to compare the data set of version i to i+1, and generate the forward difference file pair and the reverse difference match file.

(7) Then use the reverse differential server algorithm to generate the copy of the newest archiving point

i+1 and the corresponding reverse difference file pair, record the size of the forward and the reverse data.

(8) Compare the time cost: record the time cost by using the forward differential method to reconstruct the second latest archiving point files i with old files 20.1 and the corresponding forward difference file pair one by one; and record the time cost by using the reverse differential method to recover the second latest archiving point files i with the newest archiving point files i+1 and the corresponding reverse difference file pair.

(9) Repeat steps 6, 7 and 8, until version 21.2.

3.2. Experiment results

The size of the old and new file set, and the size of the forward and the reverse difference data are recorded in TABLE 1.

Table 1.The Status of Space Occupied

Version	Total size（byte）	The forward difference size（byte）	The reverse difference size（byte）	Compression ratio
old	7163385	408794	471170	6.37%
new	7481993			

It can be seen, the reverse differential algorithm on reducing the backup file space has performed very well.

We have made difference reconstruction to recover the versions of emacs. The time cost of the forward and the reverse differential method comparison shows in TABLE 2.

Table 2.Recovering Time Cost Comparision

Archiving points count N	Time cost T(s)	
	The forward differential method	The reverse differential method
1	0.109	0.343
2	0.875	0.281
3	1.406	0.281
4	1.875	0.265
5	2.281	0.266
6	2.796	0.453
7	3.593	0.328
8	4.109	0.343

Figure 3. shows more clearly about the relationship between the counts of the archiving points and the time cost to recover the second latest archiving point files.

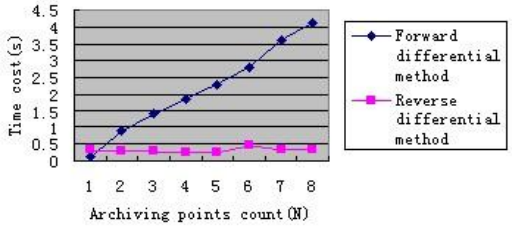

Fig. 3.Time cost versus archiving points count

From TABLE 2. and Figure 3. , we can see that with the increase of the count of the archiving point, the time cost of the reverse differential method appears to a constant to recover the second latest archiving point files, while the forward differential method appears almost linear growth. Compared to the forward differential method, the reverse differential method can significantly reduce the number of archive files reconstruction, and the time cost.

4. Conclusions

Aiming at people's habits of recovering files--usually recovering the files with recent point in time, this archiving task is achieved by calculating the forward and reverse difference set and reconstructing the latest

files with differential files by improving zdelta compression algorithm. It can significantly reduce the number of archive files reconstruction, time and resource cost.

5. References

[1] Andrew Tridgell, Paul Mackerras. The rsync algorithm[R]. Canberra: The Australian National University, 1996.

[2] Andrew Tridgell. Efficient algorithms for sorting and synchronization[D]. Canberra: The Australian National University, 1999.

[3] Dimitre Trendafilov, Nasir Memon, Torsten Suel. zdelta: An Efficieent Delta Compression Tool[R]. Technical Report TR-CIS-2002-02, Polytechnic University, CIS Department, June 2002.

[4] J. Hunt, K. P. Vo, and W. Tichy. Delta algorithms: An empirical analysis. ACM Transactions on Software Engineering and Methodology, July 1998.

[5] D. Korn and K.-P. Vo. Engineering a differencing and compression data format. In Proceedings of the Usenix Annual Technical Conference, pages 219–228, June 2002.

[6] MA Xiao-xu, HU Xiao-qin. Reverse Differential Archiving Method. Journal of Sichuan University(Engineering Science Edition), 2009.