5A-4

An alternative method for reliably managing large files

Yutaka Kawai† Adil Hasan‡ Takashi Sasaki†

1 Introduction

The volume of digital data and the size of an individual file are increasing due to the introduction of high-resolution images, high-definition audio-visual files etc. The reliable storage of such large files essentially becomes problematic with whole file replication as a failure in the integrity of the file is difficult to localise.

In this note we describe a method of managing large files in the integrated Rule Oriented Data management System (iRODS) [1, 3] by splitting them into smaller units in a traceable manner and managing the smaller units. Each unit contains its own metadata that describes its original location as well as its MD5 checksum value. We also describe the tools developed to demonstrate the method that allow the file to be split before ingestion into iRODS and assembled after extraction from iRODS. We also make use of the Resource Namespace Service (RNS) [2] to store metadata information and allow the large file to be discovered in Grid systems.

In Section 2 we describe the current reliability to manage a large file and in Sections 3 we describe the approaches to enhance the reliability. Section 4 describes some of the tests we performed in order to determine the impact of the approach and Section 5 outlines future work.

2 Current Approach To Checksum

The traditional way to checksum a large file (i.e. 50GB or so) is by preparing the checksum data for the whole file and then re-evaluating the checksum after the data has been moved [4]. Such an approach has several issues: recomputing the checksum for the whole file is time-consuming; it is not possible to identify the exact location of the discrepancy in the case of a failed checksum comparison. If we encounter checksum errors, we need to re-do the operation, or select a copy of the file that has maintained integrity. However, that is time-consuming.

Preserving Audio Visual data is required by TV and other companies today. Movie files are very large and in some cases compressed. Using the above traditional approach to checksum a file, it is not easy to figure out where the problem is or how to fix it if the downloaded data has some loss or wrong data.

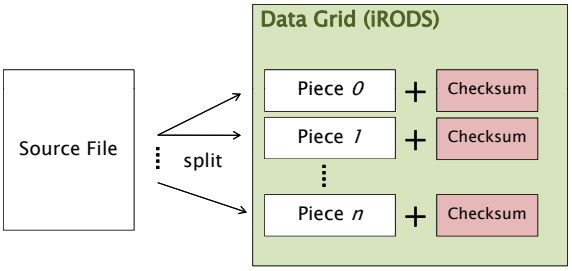

Figure 1. Splitting a file with checksum

3 Split & Checksum Approach

Our approach is to split a large file into smaller pieces, checksum each piece and assemble the file again on access. This approach can reduce the risk of loss or wrong data. Figure 1 shows how to split a file and Figure 2 shows how to combine the pieces. If we encounter checksum discrepancies, we can identify the problematic piece and retry to download the piece to resolve the discrepancy. This approach is much faster than the traditional way that is destroying and downloading the whole data.

In order to realize that, we developed a command, called 'isplit', which is able to split a large file into smaller pieces and store the pieces into iRODS with metadata. The metadata information can be also stored into RNS with xml expressions.

4 Performance Evaluation

In this section we describe the tests carried out to determine the performance impact of the overhead of checking MD5 checksum of the divided pieces. We used a 1GB file and divided it to several different numbers of pieces: 100 to

†Computing Research Center, High Energy Accelerator Research Organization (KEK)
‡School of English, University of Liverpool

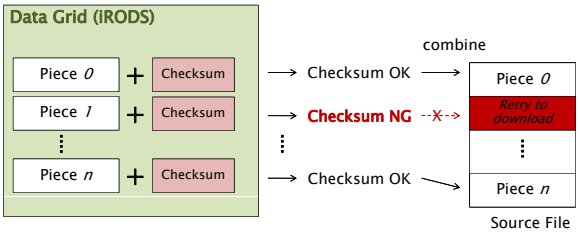

Figure 2. Combining pieces with comparing checksum

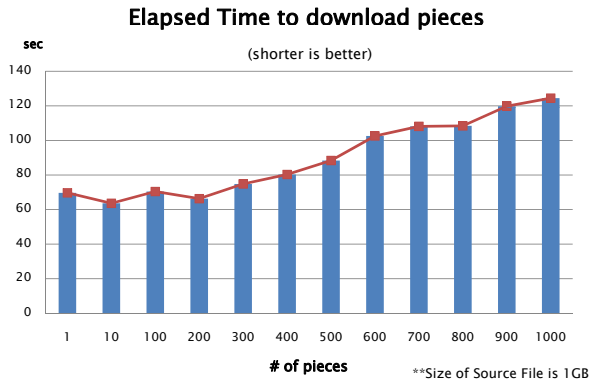

Figure 3. Speed Performance Test Results

1000 pieces. We evaluated each elapsed time to download the pieces.

4.1 Test Environment

The test environment consists of the iRODS server and the iRODS client. These machines are same and run on CentOS 5.5 as a VMware guest on the same physical machine. The physical machine has an Intel CPU i7-920@2.67GHz and 12GB RAM. The guest OS runs with 1GB RAM and one core assigned. This can have a non-trivial and noticeable effect on the results of the test.

4.2 Test Execution

We evaluated the impact of our implementations on the transfer of several different numbers of pieces: 100 to 1000. Before this demonstration, we placed all pieces on the iRODS Data Grid. Then, we download the pieces and compared each checksum data. For example, in the case of the 200 pieces, we divide the 1GB source file into 5MB pieces and placed them in the iRODS. In the demonstration, we download the all 5MB pieces and compare the checksum of each piece. We measured the elapsed time to download the pieces with comparing the checksum data. In order to get the average values, the program is executed three times for each test.

4.3 Test Results

The figure 3 shows the results of the tests. The overhead to compare the checksum data grows linearly. The case of 100 pieces is about 1.1% average slower than the non-divided case. Those cases are not so different. On the other hand, the case of 1000 pieces is about 78.6% average slower than the non-divided case which would be practically unacceptable.

The latency depends on accessing each piece to compare the checksum. As the number of pieces increases, the elapsed time increases. However, in the cases of 10, 100, and 200 pieces, the elapsed times are almost same as the non-divided case. In these cases the checksum data can be compared efficiently. Therefore, we can use this approach without a concern of speed loss when dividing a large file to less than 200 pieces.

5 Conclusion and Future work

We have shown a method for reliably managing large files. The example demonstrated that an application can split a large file and contain its MD5 checksum data as its metadata in iRODS. Also we demonstrated that the other application can compare all checksum data and then combine the pieces. We believe this approach results in more reliability with fewer replicas than in the case of large-file replication as different sub-file units can be stored on different storage systems reducing the risk due to hardware failures. We are looking at applying the approach to manage distributed files among different kinds of Data Grids.

References

[1] iRODS – the Integrated Rule-Oriented Data System. Online. `http://www.irods.org`.

[2] M. Pereira, O. Tatebe, et al. Resource namespace service specification (GFD-R-P.101). Technical report, GFS-WG, 2007. `http://www.ggf.org/documents/GFD.101.pdf`.

[3] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder. A Prototype Rule-based Distributed Data Management System. In *Proc. HPDC workshop on "Next Generation Distributed Data Management"*, Paris, France, May 2006.

[4] Z. Yong-Xia and Z. Ge. MD5 Research. 2:271 – 273, Apr. 2010.