Design and Implementation of Context-aware Hadoop InputFormat for Large-scale Scientific Dataset

Jae-Hyuck Kwak, Jun Weon Yoon, and Soonwook Hwang
Supercomputing Center
Korea Institute of Science and Technology Information (KISTI)
Daejeon, Republic of Korea
e-mail: {jhkwak,jwyoon,hwang}@kisti.re.kr

***Abstract*—Hadoop is a open-source software framework for the distributed processing of large-scale data analysis across computer clusters using a MapReduce programming model. It is becoming more popular to scientific communities including bioinformatics, astronomy and high-energy physics due to its strength of reliable, scalable data processing. Hadoop InputFormat describes the input-specification for a MapReduce job and defines how to read data from a file into the Mapper instance. Hadoop comes with several implementations of InputFormat. However, it is basically line-oriented and not suitable for context-oriented scientific data processing. In this paper, we have designed and implemented CxtHadoopInputFormat, context-aware Hadoop InputFormat for large-scale scientific dataset. Scientific dataset consists of numbers of variable-length data compartmented by user-defined context. CxtHadoopInputFormat is aware of the context in the scientific dataset and enables Hadoop to be used for distributed processing of context-oriented scientific data.**

Keywords-Data-intensive computing; Hadoop; MapReduce; Context-aware InputFormat

I. INTRODUCTION

Data-intensive computing [1] is one of the evolving scientific area which needs large-scale data analysis. Typical application areas are including bioinformatics, astronomy and high-energy physics. These scientific communities are dealing with high volume of experimental data and need reliable and scalable data management.

Hadoop [2] is a open-source software framework for the distributed processing of large-scale data analysis across computer clusters using a MapReduce programming model. Hadoop is becoming more popular to data-intensive computing application due to its strength of reliable, scalable data processing. However, it has the limitations of data processing, depending on the characteristics of scientific dataset because its default implementation is based on line-oriented and not appropriate for context-based scientific data processing.

In this paper, we have implemented CxtHadoopInputFormat, context-aware Hadoop InputFormat for large-scale scientific dataset. Scientific dataset consists of numbers of variable-length data compartmented by user-defined context. CxtHadoopInputFormat is aware of context information within scientific dataset and enables Hadoop to be used for distributed processing of large-scale scientific dataset.

The rest of this paper is as follows. We introduce Hadoop in Section 2. Then, we examine Hadoop InputFormat in Section 3, what it is and how it works. Section 4 describes the limitation of default Hadoop InputFormat implementation and the necessity of developing context-aware Hadoop InputFormat for scientific data processing. In Section 5, we describe the implementation details of context-aware Hadoop InputFormat implementation for large-scale scientific dataset. Finally, we give conclusion and future work in Section 6.

II. HADOOP OVERVIEW

Hadoop is the Apache project to develop open-source software for reliable, scalable and distributed computing. Basically, it is a framework that allows for the distributed processing of large data sets across computer clusters using a simple programming model. It can be scaled up to thousands of machines, each offering local computation and storage and delivering a highly-available service on top of that.

Hadoop consists of HDFS (Hadoop Distributed FileSystem) and MapReduce. HDFS is a distributed file system, which is highly fault-tolerant and provides high throughput access to application data.

Figure 1 shows HDFS architecture. HDFS cluster consists of a single NameNode and a number of DataNodes.

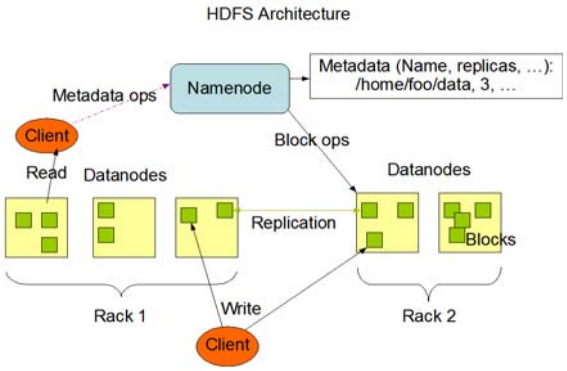

Figure 1. HDFS Architecture

A File in HDFS is split into one or more blocks and these blocks are stored and replicated in a set of DataNodes. NameNode executes file system namespace operations like opening, closing, and renaming files and directories and determines the mapping of block to DataNodes. DataNode is responsible for serving read and write request requests from the file system's clients and performs block creation, deletion, and replication under the NameNode's instruction.

MapReduce is a simple programming model for processing and generating large data sets. It consists of Map and Reduce functions, respectively.

Map(key1, value1) → list<key2, value2>
Reduce(key2, list<value2>) → list<value3>

Figure 2 shows MapReduce dataflow. A map function transforms input data row of key and value to an intermediate output key/value. A reduce function take all values for a specific key, and generate a new list of the final output.

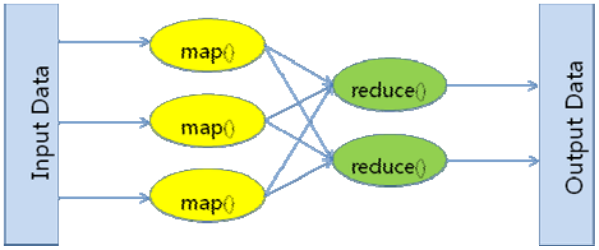

Figure 2. MapReduce Dataflow

The key advantage of the MapReduce is that every Map and Reduce is independent of all other ongoing Maps and Reduces, then the operation can be run in parallel on different keys and lists of data. If you write an application in the MapReduce form, scaling the application to run over hundreds of machines in a cluster is merely a configuration change. Furthermore, Hadoop runs the Map functions on compute nodes where the data lives, rather than copy the data over the network to the program. The output list can then be saved to HDFS and the Reduce functions run to merge the results.

Hadoop is suitable for applications which have large data sets due to HDFS block size (64MB by default) and in-memory representation of the filesystem metadata. Applications that run on HDFS need streaming access to their data sets and write-once-read-many (WORM) access model for files. As a result, Hadoop is suitable for applications which need high-throughput access of large-sized data.

III. HADOOP INPUTFORMAT IMPLEMENTATION

The InputFormat describes the input specification for a MapReduce job and defines how to read data from a file into the Mapper instances. MapReduce relies on the InputFormat of the job to:

1. Validate the input-specification of the job
2. Split-up the input files into logical InputSplits, each of which is then assigned to an individual Mapper.
3. Provide the RecordReader implementation to be used to glean input records from the logical InputSplit for processing by the Mapper.

Main goal of the InputFormat is to divide the input data into fragments that make up the inputs to individual map tasks. These fragments are called "splits" and are encapsulated in instances of the InputSplit interface. Most files are split up on the boundaries of HDFS block size, and are represented by instances of the FileInputSplit class. RecordReader ensures that the splits do not necessarily correspond neatly to line-ending boundaries and do not miss records that span InputSplit boundaries.

Hadoop comes with several implementations of InputFormat. TextInputFormat is the default InputFormat that each record is a line of input. Within TextInputFormat, the key is the byte offset within the file of the beginning of the line and the value is the contents of the line. NLineInputFormat receives a fixed number of lines of input. Like TextInputFormat, the keys are the byte offsets within the file and the values are the lines themselves.

IV. IMPLEMENTATION OF CONTEXT-AWARE HADOOP INPUTFORMAT

Hadoop InputFormat handles input data formats, how it handles the way input data is split into parts for processing by the map tasks, and how it handles the extraction of atomic data from the split data. Figure 3 describes default InputFormat implementation.

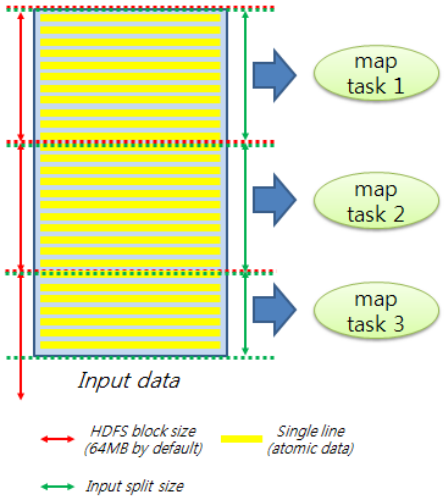

Figure 3. Default Hadoop InputFormat Implementation

 ISBN: 978-1-61208-226-4

In default InputFormat implementation, the atomic data is mostly a line of text (bold yellow line) in a file separated by carriage returns. Hadoop normally processes a very large data file containing a long sequence of atomic data that each can be processed independently. The file is split automatically, normally along HDFS block boundaries (dotted red line) and each split data (dotted green line) is passed to the separate map task for processing.

However, the situation in the scientific data processing could be different in two aspects at least. First, atomic data could be not only a line, but also a multi-line block, the rows of a DB table or a file in the folder. Second, the split data should be made on the boundary of atomic data. It could be not only end of line, but also end of multi-line block, end of row records or end of a file. These aspects are depending on the characteristics of scientific data processing application.

Default InputFormat implementation has some limitations to accommodate this situation and could result in a incorrect data processing. Figure 4 describes a broken context in default InputFormat implementation. In this example, multi-line block (yellow box) should be processed atomically by the map task. However, default InputFormat implementation does not know about this and try to split the data in the middle of atomic data, which is on the boundary of HDFS block. As a result, the data context in the file could be broken.

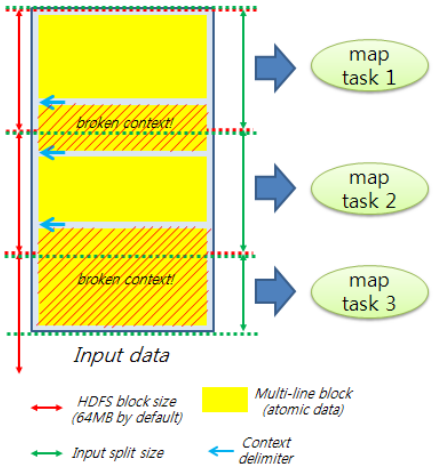

Figure 4. Broken context in default Hadoop InputFormat Implementation

To overcome this situation from scientific data processing on Hadoop, we have implemented context-aware Hadoop InputFormat. Section 5 describes the implementation details about this.

V. IMPLEMENTATION OF CONTEXT-AWARE HADOOP INPUTFORMAT

Scientific dataset, especially from large-scale experiment-oriented science such as high-energy physics, astronomy and bioinformatics mostly consists of unstructured data. For example, high-energy physics uses distributed Monte-Carlo simulation and outputs data files what you got from the experiment. Astronomy deals with observation data files which is extracted from the observation instruments.

Hadoop is suitable for large-sized data processing. Scientific dataset could be merged depending on scientific data processing framework. Typical data structure comes with numbers of variable-length data compartmented by user-defined context, depending on the data structure from the field of science application. As mentioned in Section 2, though Hadoop can be used for reliable, scalable and distributed processing of large-scale scientific dataset, default InputFormat implementations are line-oriented, not context-oriented. MapReduce framework reads data from computed input splits and assigned to the Mapper. However, input split is calculated by the formula of file size and HDFS block size, not taking into account the data context in the file. This is likely that file can be split at the wrong position and then cause the incorrect result.

We have implemented CxtHadoopInputFormat, context-aware Hadoop InputFormat. Figure 5 shows CxtHadoopInputFormat implementation.

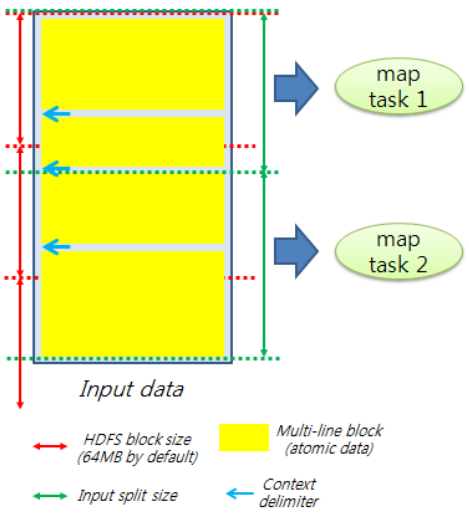

Figure 5. Context-aware Hadoop InputFormat Implementation

We assume that scientific dataset has the so-called “context delimiter” that compartmentalized into input records on per-context basis. Our implementation is aware of context delimiter and ensures that input data are split up on the boundaries of the context delimiter, not on the boundaries of HDFS block size.

Figure 6 shows CxtHadoopRecordReader class which implements RecordReader interface. We reuses LineRecordReader class which is the RecordReader implementation used by TextInputFormat. We wrapped the LineRecordReader with our own implementation which converts the value to the expected types.

 ISBN: 978-1-61208-226-4

The next method is called repeatedly to populate the key and value objects. It reads input split in lines and add them to value object until line is about to start by context delimiter. The next method returns value object which contains input records that have the same data context.

```
public class CxtHadoopRecordReader implements RecordReader<LongWritable, Text> {
    …
    public synchronized Boolean next(LongWritable key, Text value) throws IOException {
        // read input split in lines
        // add line to value object until line is about to start by context delimiter
    }
}
```

Figure 6. CxtHadoopRecordReader class

Figure 7 shows CxtHadoopInputFormat class that extends FileInputFormat class. We need to define a factory method for RecordReader implementations to return new instance of CxtHadoopRecordReader class.

```
public class CxtHadoopInputFormat extends FileInputFormat<LongWritable, Text> implements JobConfigurable {
    …
    public RecordReader<LongWritable, Text> getRecordReader(InputSplit genericSplit, JobConf job, Reporter reporter) throws IOException {
        reporter.setStatus(genericSplit.toString());
        return new CxtHadoopRecordReader(job, (FileSplit) genericSplit);
    }
    …
    Public InputSplit[] getSplits(JobConf job, int numSplits) throws IOException {
    // identify hostname, offset and size of data
    // read file in lines
    // generate new input split if line is started by context delimiter and larger than user-defined splitSize

    }
}
```

Figure 7. CxtHadoopInputFormat class

The getSplits method gets the desired number of map tasks as the numSplits argument. This number is treated as a hint and a different number of input splits can be made. In our implementation, The getSplits method reads input file from HDFS and calculates input splits on the boundaries of context delimiter if the size of input split is larger than user-defined splitSize.

VI. CONCLUSION AND FUTURE WORK

Hadoop is one of the emerging technologies for large-scale data analysis. However, it has some limitations to deal with context-oriented scientific dataset. In this paper, we have implemented context-aware Hadoop InputFormat for processing context-oriented scientific dataset using the Hadoop. CxtHadoopInputFormat is aware of the context in the scientific dataset and enables Hadoop to be used for distributed processing of context-oriented scientific data by spliting up the input data on the boundaries of the context in the scientific dataset correctly.

We have a plan to apply CxtHadoopInputFormat to the representative scientific data processing. We are working with astronomy scientists who want to process data files from SuperWASP project, which is the UK's leading extra-solar planet detection program. They have quite many data files extracted from the observation cameras and are willing to processs them on Hadoop framework to know how it can help large-scale scientific data processing from the field of astronomy. CxtHadoopInputFormat will be helpful to deal with SuperWASP dataset on Hadoop framework.

REFERENCES

[1] Ian Gorton, Paul Greenfield, Alex Szalay, and Roy Williams, Data-Intensive Computing in the 21st Century, Computer, vol. 41, Apr. 2008, pp. 30-32, doi:10.1109/MC.2008.122.

[2] Apache Hadoop Project, http://hadoop.apache.org

[3] Yahoo! Hadoop Tutorial, http://developer.yahoo.com/hadoop/tutorial/

[4] Tom White, Hadoop: The Definitive Guide (2nd Ed), Oreilly, 2011.

[5] Chuck Lam, Hadoop in action, Manning, 2010.

[6] Jason Venner, Pro Hadoop, Apress, 2009.

[7] Hadoop 0.20.2 API, http://hadoop.apache.org/common/docs/r0.20.2/api/

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google file system," Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP'03), Oct. 19-22, 2003.

[9] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters," Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation (OSDI'04), pp. 10-10, Dec. 06-08, 2004.

[10] Hadoop Performance Tuning, Impetus White Paper, http://www.impetus.com

[11] Milind Bhandarkar, Suhas Gogate, and Viraj Bhat, Hadoop Performance Tuning A case study, http://cloud.citris-uc.org/system/files/private/BerkeleyPerformanceTuning.pdf.

 ISBN: 978-1-61208-226-4