HITACHI
Inspire the Next

Big Data – XML Parsing in Pentaho Data Integration (PDI)

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

Overview 1

Before You Begin 1

Terms You Should Know 1

Selecting the Best Method Based on Use Case 3

XML Files Over 128MB (Default Block Size) 3

Many XML Files Smaller than 128MB Each 3

A Few Small XML Files 3

Methods Implementation and Details 4

Mahout `XMLInputFormat` 4

File List Processing in Pentaho MapReduce 5

File List Processing in PMR (Small Set) 7

Convert XML to Binary Format 7

Write a Custom Input Formatter 8

Store the Entire XML as a Single Line 8

Use HBase to Store the Entire XML in a Column 9

Related Information 9

This page intentionally left blank.

Overview

This document describes different techniques to process and parse Extensible Markup Language (XML) files stored in a Hadoop cluster. This best practice focuses on selecting and implementing the best strategy for your use case.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

Software	Version(s)
Pentaho	7.x, 8.0

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document.

Terms You Should Know

Here are some terms and concepts you should be familiar with:

- **Hadoop File Storage**: Hadoop file storage (HDFS) is based on a distributed file system. HDFS is structured similarly to a regular Unix file system, but its data storage is distributed across several machines. It is intended as a file system layer for use by large distributed systems, not as a replacement for a regular file system. It has built-in mechanisms to handle machine outages, and is optimized for throughput, rather than latency.
  HDFS addresses big data challenges by breaking files into a family of smaller blocks, which are distributed among the data nodes in the HDFS cluster and managed by the `NameNode`. HDFS block sizes are configurable and default to 128MB, but can be set higher. Therefore, by default, a 1GB file consumes eight 128MB blocks for its basic storage needs.
  Reflecting HDFS's resiliency, the blocks are replicated throughout the cluster in case of a server failure, with a default replication factor of three. Apache's HDFS Architecture Guide has more in-depth background information about HDFS.

- **MapReduce**: HDFS and MapReduce perform their work on nodes in a cluster hosted on racks of commodity servers. When a client requests a MapReduce program be run, the first step is locating and reading the input file containing the raw data. The file format is arbitrary, but the data must be converted to something that the program can process. `InputFormat` performs this function and determines how the file will be broken into smaller pieces for processing, using a function called `InputSplit`.

`InputFormat` then assigns a `RecordReader` to transform the raw data for processing by the map. Several types of `RecordReader` are supplied with Hadoop, offering a wide variety of conversion options. This feature is one of the ways that Hadoop manages the large number of data types found in big data problems. You can find more details in this Hadoop MapReduce Tutorial.

- **How HDFS Design Affects XML File Types**: The HDFS default block size is 128MB, but can be set higher based on the use case. Large XML files are split by the HDFS-configurable block sizes parameter, `dfs.blocksize`.
  When XML file sizes are small and do not require splitting, HDFS and MapReduce can generate overhead on reading small files sequentially.
  MapReduce defaults to creating a Java virtual machine (JVM) process called a reader container for each file, although there are settings you can change to prevent this.

Selecting the Best Method Based on Use Case

This section covers a few different approaches and methods to administer, ingest, or process your XML files in a heavy-duty XML system, depending on the number of XML files that need to be processed. The following cases describe recommended approaches and different options to consider for each.

You can find details on these topics in the following sections:

- XML Files Over 128MB (Default Block Size)
- Many XML Files Smaller than 128MB Each
- A Few Small XML Files

XML Files Over 128MB (Default Block Size)

You will need to split files larger than the default HDFS block size of 128MB in the cluster among multiple nodes.

> *If you have a 3GB file with a 128MB block size, you will have 24 blocks of the file that are distributed in the cluster with proper replication.*

In such a case, use `XMLInputFormat` (Mahout Project) so that the mappers run on the block on the node where data is stored (data locality). This method, the standard Hadoop use case solution, results in minimal memory consumption and fast processing speed. More information is available in this document in the Mahout `XMLInputFormat` section.

Many XML Files Smaller than 128MB Each

If you have many XML files smaller than 128MB, `XMLInputFormat` will slow down performance. This happens because a mapper needs to be started for each file, and it generates a lot of overhead that can be avoided.

There are a few different approaches that you can use to address this, described in the Methods Implementation section.

A Few Small XML Files

If you have a few small XML files, you can use a direct method. You could push all the work to be done by the Pentaho Data Integration (PDI) Server, or use a single mapper approach. There is no need to list the files or distribute them to the mapper. Instead, the mapper can be written in such a way that it reads the source and then writes the output. More details can be found in the File List Processing in PMR (Small Set) section of this document.

Methods Implementation and Details

This section covers methods for processing XML data stored in HDFS, using Mahout and code implementation, file list processing in MapReduce, which helps distribute sets of lines to mappers, and ways to convert XML to binary format.

Mahout *XMLInputFormat*

Apache Mahout is a set of libraries that provide machine learning at scale. Part of the function of these libraries is to include input formatter implementation that helps process XML data stored in HDFS.

Code implementation for processing large XML files in Hadoop with MapReduce proceeds in this way:

1. Hadoop stores the large XML files in many blocks which are distributed and replicated around the cluster. The block size is configurable, but the default value can vary based on distribution. For example, Cloudera's default block size is 128MB.
2. Distributed blocks are formed of XML line elements that do not analyze or keep integrity of the XML per block. Some logic needs to be implemented to process large XML in distributed systems.
3. Mahout implementation requires defining an XML tag that represents the repetitive set of information that needs to be extracted from large XML files. Mahout's set of libraries will guarantee that each mapper receives the full content of the XML node surrounded by the defined tag.

Take the following XML file, for example:

```
<root>
	<headerTag>
		<content></content>
	</headerTag>
	<data>
		<date>1/1/2004</date>
		<name>name1</name>
		<description></description>
	</data>
	<data>
		<date>2/2/2017</date>
		<name>name2</name>
		<description></description>
	</data>
	<data>
		<date>3/5/2000</date>
		<name>name3</name>
		<description></description>
	</data>
</root>
```

If the tag for `data` is selected, Mahout libraries will make sure each mapper receives a full set of `data` tags.

Remember that the `headerTag` information, or other tags, will be ignored by the libraries:

```
<data>
	<date>1/1/2004</date>
	<name>name1</name>
	<description></description>
</data>
```

`data` XML content will be retrieved as rows of information like this:

```
<data><date>1/1/2004</date><name>name1</name><description></description></d
ata>
<data><date>2/2/2017</date><name>name2</name><description></description></d
ata>
<data><date>3/5/2000</date><name>name3</name><description></description></d
ata>
```

Each mapper will receive `data` tag elements based on data locality; therefore, even if each chunk of data has a natural order, the order between these elements can neither be determined nor controlled. You will have to take additional steps if you require ordering.

Table 1: Advantages and Disadvantages of Mahout `XMLInputFormat`

Advantages	Disadvantages
Inbuilt parser	• Not intuitive • Implementation challenges

File List Processing in Pentaho MapReduce

This method uses the `NLineInputFormat` to distribute sets of lines to the mappers and assign nodes based on availability and resources instead of data locality.

This technique is strong when the action to be performed requires more computation activity per list item (line). Pentaho's engine is started on each mapper to perform any of the Pentaho component's functionality. Normally, this method is used in mappers only, and the use of reducers is optional. Processes defined using this methodology can be implemented in one of two ways:

- Limit the number of mappers (parallel processes) that can be assigned to do an action or task.
- Limit the number of action items (at a time, or during a period of time) to be processed by one mapper or worker.

This method is based on `NLineInputFormat` creating a text file with the list of filenames (full path) that the ETL needs to read. Filenames are processed with one filename path on each line, and then Pentaho MapReduce (PMR) reads each line individually from the list of files.

To do this, use the `org.apache.hadoop.mapred.lib.NLineInputFormat` included in the Hadoop distribution. This format receives `mapreduce.input.lineinputformat.linespermap`, which is a parameter with the number of fixed lines that will be sent to each mapper.

The `NLineInputFormat` library will see the number of lines in the input file and queue all the chunks of data (lines per mapper) that will be required. The node assigned by the resource manager will be based on available capacity and not data locality.

Hadoop monitors whether the mappers are reading or writing data, and may terminate a mapper if it detects no activity in the mapper during the default wait time of 60 minutes. To change this, you can use custom readers or writers in place of the MapReduce ones, but you may need to add this parameter if you want unlimited wait time:

```
mapreduce.task.timeout=0
```

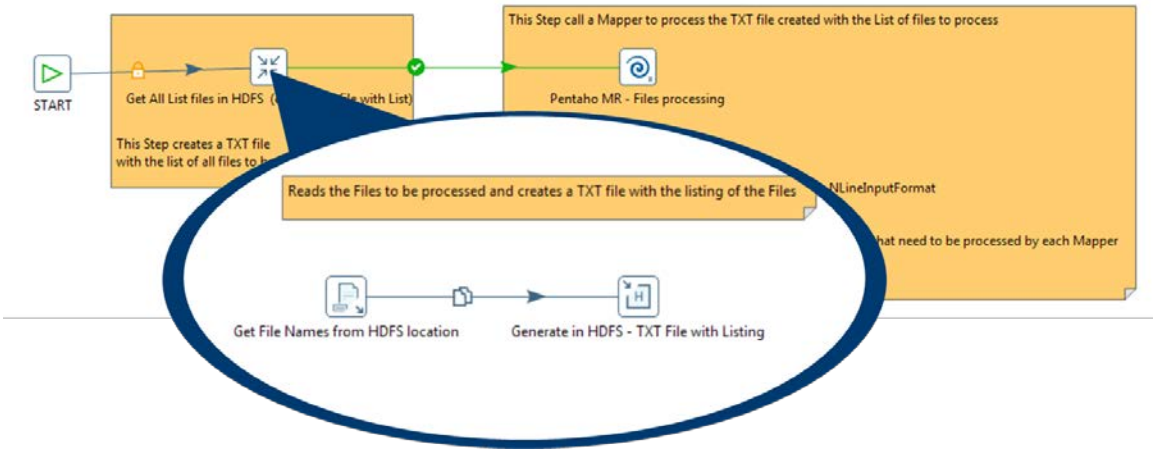

Figure 1: Main Job - Create File List

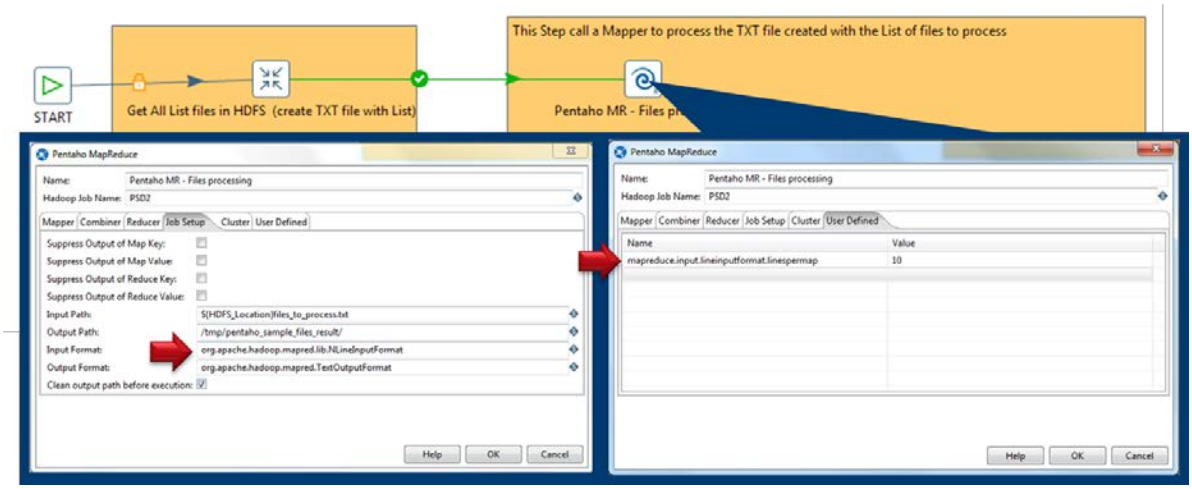

Figure 2: Main Job - Call PMR with `NLineInputFormat`

Table 2: Advantages and Disadvantages of PMR

Advantages	Disadvantages
XML files can be processed without any preprocessing.	Mappers process the files without data locality. Other `InputFormats` that extend this idea and include data locality are available from the community.

File List Processing in PMR (Small Set)

This process is a variation of the above File List Processing in Pentaho MapReduce method, and works for lists of files that are small enough to be fully processed by one mapper in the Hadoop cluster. It will generate the text file with the list of files, and distribution on the cluster then becomes an overhead.

The parent job creates a single-line input file that can be reused by multiple implementations. Paired with `NLineInputFormat`, this creates a single mapper on the cluster. That mapper will not use the MapReduce `INPUT` data, but will instead use the **Get data from XML** step with reference to a static or dynamic HDFS location with the `RegEx` pattern to read all the files.

This is the same method used in regular XML processing within non-Hadoop processes.

Table 3: Advantages and Disadvantages of File List Processing in PMR (Small Set)

Advantages	Disadvantages
• XML files can be processed without any preprocessing. • Simplifies the creation of transformations.	• Mappers process the files without data locality. • Only works for manageable number of files that can be processed by one mapper.

Convert XML to Binary Format

This method collects and transforms the small XML files to a large text file that collects all the information. This way, we convert the small files overhead into a big data-oriented process.

First, all the small XML files are collected and converted to binary format, putting the data in one line without losing real format or character returns. Next, the data is appended to a file that can be used later for any processing. This requires a two-step process; however, the conversion of source XML files to binary can be done at ingestion time instead of at processing time.

Figure 3: Converting from XML to Binary

After the file is created, it can be processed with PMR and `TextInputFormat`. Once you have converted the file from binary to text, each line in the file will have all the XML content that you will need to process with the **Get data from XML** step.

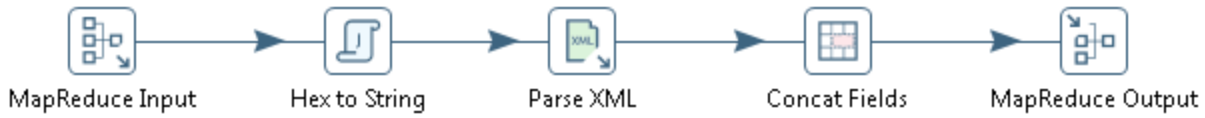

Figure 4: PMR Mapper Illustration

Both processes can be done in one orchestration job, as shown in Figure 5:

Figure 5: Orchestration Job for Converting XML to Binary and Processing

Table 4: Advantages and Disadvantages of Converting XML to Binary and Processing

Advantages	Disadvantages
You can: • Split input • Use multiple mappers • Use out of the box PDI capabilities	Must convert twice: • XML to binary when storing • Binary to string when processing in MapReduce

Write a Custom Input Formatter

This method requires Java skill to write a custom input formatter, which must read the complete file content as one record. In addition, the formatter must use a custom `RecordReader` class, and `isSplitable` must return `false`.

Table 5: Advantages and Disadvantages of Writing a Custom Input Formatter

Advantages	Disadvantages
Popular and customizable if you know Java	Overheads of working with external Java code

Store the Entire XML as a Single Line

This method is like Convert XML to Binary Format, where all the individual files and XML blocks are linearized to guarantee that any split of the blocks in the HDFS retains line-by-line integrity. This way, you can still read the file per line to record and process it with the **Get data from XML** step.

Table 6: Advantages and Disadvantages of Storing the Entire XML as a Single Line

Advantages	Disadvantages
You can: • Combine many small XML files into one • Use many mappers	You may lose line breaks from data.

Use HBase to Store the Entire XML in a Column

This method, for small and manageable XML file sizes, uses HBase to store the full XML content in a column. Once the data is in HBase, you can read and process it with the PDI **HBASE Row Decorator** step.

Table 7: Advantages and Disadvantages of Using HBase to Store the Entire XML in a Column

Advantages	Disadvantages
Use multiple mappers, one per column.	• Overhead of maintaining HBase Data stored as a byte array • Not human-readable

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- Apache's HDFS Architecture Guide
- Apache Mahout
- Hadoop MapReduce Tutorial
- HDFS Default Block Size
- Pentaho Components Reference