HITACHI
Inspire the Next

Transformation Variables in Pentaho MapReduce

Contents

Overview 1
Before You Begin 1
Prerequisites 1
Use Case: Side Effect Output 1
Pentaho MapReduce Variables 2
Injected Variables 2
Using the TaskID for Side Effect Outputs 2
TaskID in PMR Transformations 2
Example TaskID String 3
Example Transformation Using TaskID 3
Related Information 3
Finalization Checklist 4

Overview

This document covers some best practices on Pentaho MapReduce (PMR), which allows extract/transform/load (ETL) developers to design and execute transformations that run in Hadoop MapReduce. When PMR jobs are performed, additional variables are injected into the Kettle variable space that can be used to enhance transformations that map, combine, or reduce.

Our intended audience is Pentaho Data Integration (PDI) developers or administrators configuring PMR on a Hadoop cluster.

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

Software	Version(s)
Pentaho	6.x, 7.x, 8.x

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.

Prerequisites

This document assumes that you have knowledge of Pentaho and that you already have connectivity to a Hadoop cluster. More information about related topics outside of this document can be found in the Pentaho MapReduce documentation.

Use Case: Side Effect Output

Fabiola wants to use a file output not specifically defined by her job's FileOutputFormat. This means she will need to use a side effect output and will have to clean it up manually, since it will not be automatically managed by the MapReduce framework.

Pentaho MapReduce Variables

All PMR jobs are launched by an entry in a PDI job. When the PMR entry is executed, the mapper, reducer, and combiner transformations inherit the variable space of the entries' parent job. This variable space is then communicated to the transformations through the MapReduce job configuration.

However, there are some additional variables that get injected into the MapReduce transformations, and these variables can be useful for PMR developers.

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

- Injected Variables
- Using the TaskID for Side Effect Outputs

Injected Variables

These are the variables that are injected into the PDI transformation variable space from the Hadoop mapper, combiner, or reducer context:

- `Internal.Hadoop.NumMapTasks` is the number of map tasks from the MapReduce job configuration.
- `Internal.Hadoop.NumReduceTasks` is the number of reducers configured for the MapReduce job. If the value is `0`, then a map-only MapReduce job is being executed. Use positive integers in this variable for key partitioning design from map tasks.
- `Internal.Hadoop.TaskId` is the `taskID` of the mapper, combiner, or reducer attempt context. This variable contains the MapReduce or Yarn attempt task string, which is useful for side effect outputs.

Using the TaskID for Side Effect Outputs

When you use map or reduce transformations, you may want to use file outputs aside from the output specifically defined by your job's `FileOutputFormat`. These outputs, created from map or reduce contexts, which are not from the job's output format, are called **side effect** outputs.

When a MapReduce job fails, or a task in a MapReduce job fails, the MapReduce framework cleans up any output written through the job's configured output format, but **side effect** files are not automatically managed by the framework. Instead, they must be cleaned up manually.

TaskID in PMR Transformations

The `taskID` is an excellent string to use when you are building a unique filename for side effect outputs.

You may want to record additional debug or error data outside of the MapReduce output format. Although you could use a unique `userID` (UUID) to generate a unique name, using the `taskID` provides you with information that can be tracked back to a specific mapper or reducer in the MapReduce job history.

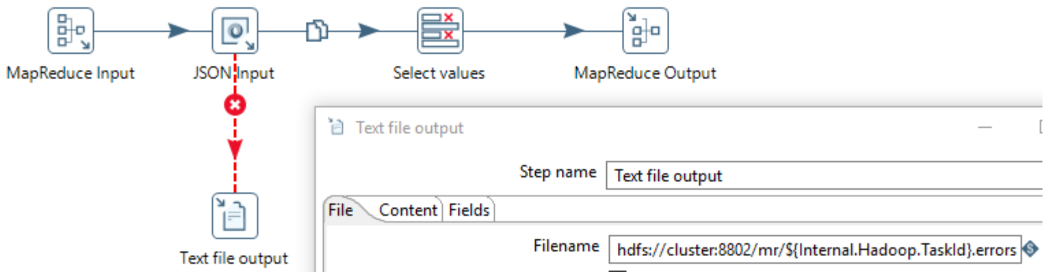
Example TaskID String

An example `taskID` string may look like this:

```
attempt_1507840814927_0015_r_000002_0
```

Apache does not recommend you parse this string, but you can tell just by looking at it what the `jobID` is (`1507840814927_0015`), `taskType` (`r`, for reducer), `partitionNumber` (`000002`), and `attemptID` (`0`).

Example Transformation Using TaskID



Step name: Text file output
Filename: hdfs://cluster:8802/mr/${Internal.Hadoop.TaskId}.errors

Figure 1: MapReduce Transformation

In the figure, the **MapReduce Input** step reads JavaScript Object Notation (JSON) strings, which get parsed by the **JSON Input** step.

If there are errors parsing input strings in the **JSON Input** step, error rows get directed to the **Text file output**, whose output filename is configured as a VFS location on the cluster. That location includes the `Internal.Hadoop.TaskId` variable, to create a unique file capturing the errors from the mapper/reducer context of this PMR job.

Rows that are successfully parsed in the **JSON Input** step move on to the **Select values** and **MapReduce Output** steps, where the rows would be put out using the job's configured output format.

Related Information

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

- Apache MapReduce
- Class TaskID
- Pentaho Components Reference
- Pentaho MapReduce

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project:____

Date of the Review:____

Name of the Reviewer:____

Item	Response	Comments
Did you utilize the `Internal.Hadoop.TaskId` variable to generate a side-effect file?	YES________ NO________	
Have you established a process to cleanup any side-effect files generated?	YES________ NO________	