pentaho®
A Hitachi Group Company

# Best Practices - Big Data On-Cluster Processing

## Contents

## Overview

This document covers best practices to push ETL processes to Hadoop-based implementations. It is important that we consider concepts like data locality, distributed processing, resource assignment isolation, and processing queue among other topics, when working on big data implementations.

The Pentaho Data Integration (PDI) tool includes multiple functions to push work to be done on the cluster using distributed processing and data locality acknowledgment. However, it is important to understand what the use cases and situations are when the use of a method is recommended or not.

*The PDI Server is referred to as the ETL Server beginning in Pentaho 7.0. This document applies to both.*

The following items are covered in this document:

- Pentaho Deployment Architecture Recommendations
- Current On-Cluster Processing Options

| Software | Version |
|---|---|
| **Pentaho** | 5.4, 6.x, 7.x |
| **Hadoop** | Cloudera, Hortonworks, MapR |

## Pentaho Deployment Architecture Recommendations

The PDI or ETL server needs to communicate with the Hadoop Distributed File System (HDFS) name nodes and data nodes, resource manager, Hive server, Impala server, and Oozie, based on the activities to be performed by the ETL designed process.

Visit the following links for more information regarding topics discussed here:

For Pentaho 5.4 or 6.x:

- Pentaho BA Server and High Availability
- Pentaho DI Server and High Availability

For Pentaho 7.x:

- Pentaho Servers with High Availability
- Pentaho ETL Servers with High Availability

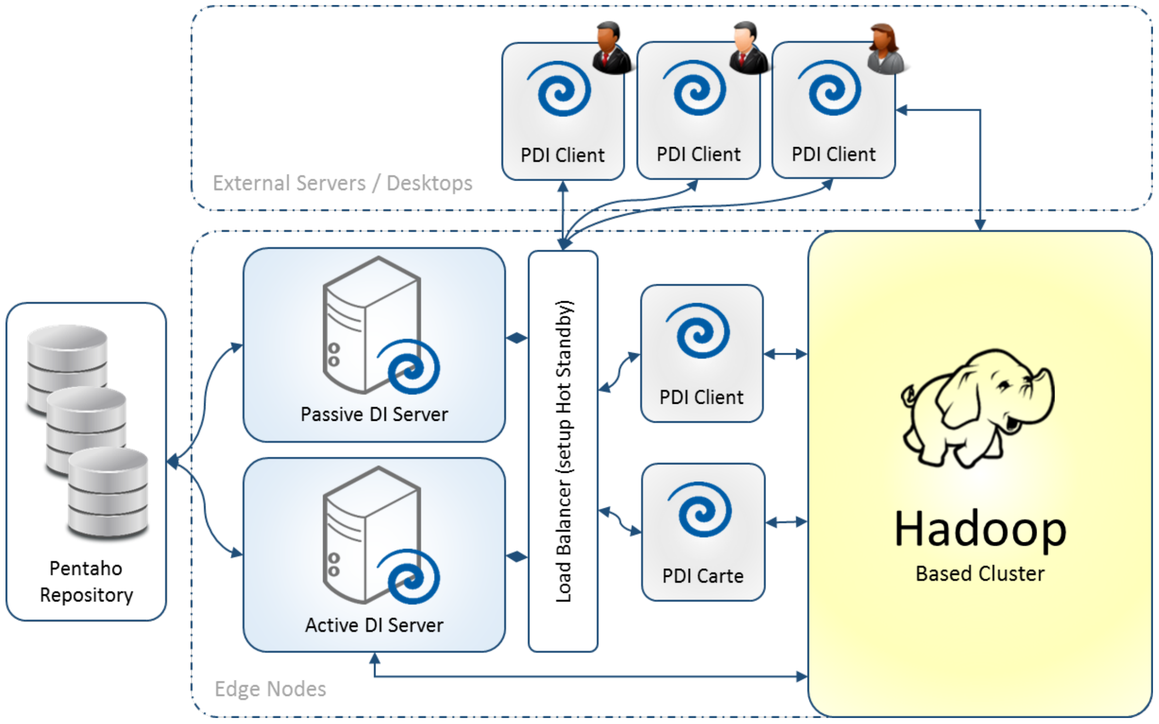

*Figure 1: PDI Server Deployment*

The diagram above shows the importance of highlighting PDI clients installed in the Edge nodes. There are also PDI clients outside the cluster nodes that have network access to the clusters.

For more information about topics discussed here, see our Best Practice documentation on Pentaho Big Data Integrated Authentication.

## Hadoop and PDI Integration

Pentaho MapReduce relies on Hadoop's distributed cache to distribute PDI's libraries and plugins across the cluster. PDI will upload all its dependent JARs to HDFS in the following target directory when you initially run a PDI application on a Hadoop cluster:

```
hdfs://your.hdfs-host.com:port/opt/pentaho/mapreduce/[pdi-
version-hadoop-version]
```

PDI will not upload anything if the target HDFS directory already exists. The HDFS directory is set in the `plugin.properties` files with a parameter of `pmr.kettle.dfs.install.dir`. The `plugin.properties` file for individual Pentaho applications are listed in the following table:

*Table 1: Target Plugin Properties*

| Target | Plugin Property |
|---|---|
| **(Pentaho 7.x) Pentaho Server** | `[PENTAHO_INSTALL_HOME]/server/pentaho-server/pentaho-solutions/system/kettle/plugins/pentaho-big-data-plugin/plugin.properties` |
| **DI Server** | `[PENTAHO_INSTALL_HOME]/server/data-integration-server/pentaho-solutions/system/kettle/plugins/pentaho-big-data-plugin/plugin.properties` |
| **BA Server** | `[PENTAHO_INSTALL_HOME]/server/biserver-ee/pentaho-solutions/system/kettle/plugins/pentaho-big-data-plugin/plugin.properties` |
| **Report Designer** | `[PENTAHO_INSTALL_HOME]/design-tools/report-designer/plugins/pentaho-big-data-plugin/plugin.properties` |
| **Spoon/Spoon/Pan/Kitchen** | `[PENTAHO_INSTALL_HOME]/design-tools/data-integration/plugins/pentaho-big-data-plugin/plugin.properties` |
| **Metadata Editor** | `[PENTAHO_INSTALL_HOME]/design-tools/metadata-editor/plugins/pentaho-big-data-plugin/plugin.properties` |

*We recommend giving PDI user permission to write to the root HDFS directory when setting up the HDFS.*

You can manually create the `/opt/pentaho` directory and give PDI user write permission. You can also change the `pmr.kettle.dfs.install.dir` parameter to a location with write permission. See the `plugin.properties` file for more details about the directory structure.

PDI will also allow you to run multiple versions on the same Hadoop cluster. This will allow you to run different versions of PDI on the same Hadoop cluster to test for PDI upgrades.

*Be sure to delete the `pmr.kettle.dfs.install.dir/[pdi-version-hadoop-version]` directory in HDFS if you change PDI's dependent JARs by using a patching process, or apply JAR replacements where the Hadoop version or PDI version does not change. This will allow the new JARs to be uploaded to HDFS during the next run of a Pentaho MapReduce.*

# Current On-Cluster Processing Options

This section provides information about the orchestration process and demonstrates how to send work to the cluster. It also provides information about running transformations on the cluster, and tips for writing better PDI jobs and transformations.

- Orchestration
- Run Transformations on the Cluster
- Pentaho MapReduce
- YARN Cluster
- Tips for Writing Better PDI Jobs and Transformations

## Orchestration

PDI offers multiple ways to push work to the cluster. Some of these options are agnostic tools. PDI works as an orchestrator that calls cluster processes. It then waits for a response and coordinates the next steps based on a `SUCCESS` or `FAIL` result. This method does not have control or influence on how the cluster executes the called process. However, Pentaho provides the cluster with the user credentials for any resource isolation configuration that is in place for the calling user.

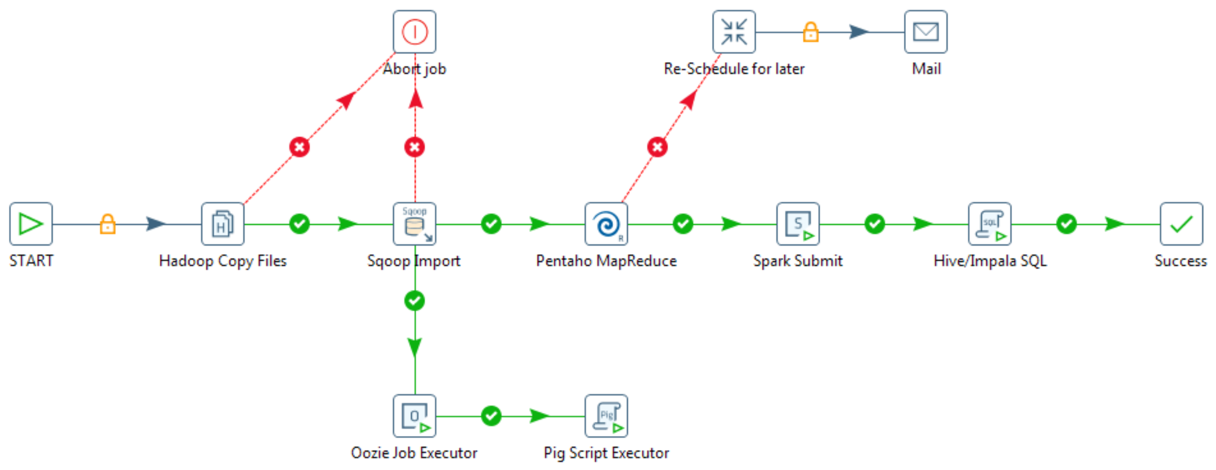

*Figure 1: PDI Orchestration*

The orchestration process runs on the server or node where the PDI process was started. The machine will orchestrate the activities if the job is executed from an end-user or developer machine. Any shutdown or kill of the application on the machine will stop the job submitted.

## *Pentaho MapReduce*

Pentaho MapReduce job steps wrap PDI transformations into the Hadoop MapReduce architecture. The transformations are classified as mapper transformation and reducer transformations.

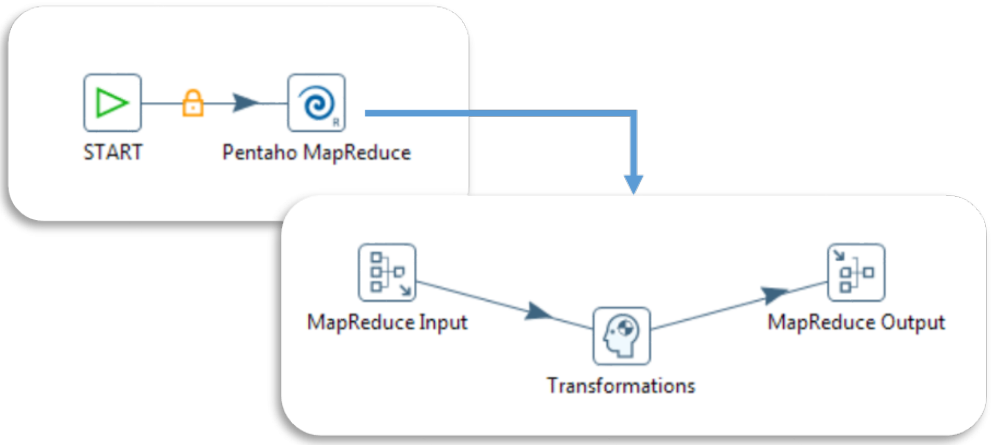

*Figure 2: Weblog File Parsing*

A collection of resources that show standard and common usage of the MapReduce functionality can be found in the following links:

- Pentaho MapReduce Workflow
- Create Mapper and Reducer Aggregate Dataset
- Create Mapper Transformation to Parse Weblog File

## *Best Practice Recommendations with MapReduce*

**Pentaho MapReduce Job Steps:**

1. You should check the **Enable Blocking** options in the cluster tab if you want to wait for the MapReduce process to complete. If you do not, the MapReduce process will start in Hadoop and return as complete.
2. Hadoop may ignore the number of mappers that you want to run. However, you can view the number of reducers on Pentaho's Hadoop Wiki.
3. Hadoop will not run the shuffle and sort phase if you set the number of reducers to zero. The unsorted mapper output will be moved to the configured output directory.
4. The shuffle and sort phase will be run using the Identity Reducer. This will happen if you do not provide a reducer implementation, and if the number of reducers is not set or is greater than zero. This will cause the mapper output to be shuffled and sorted and output to the configured directory.

**Hadoop Input Steps:**

1. Withdraw the data from the HDFS and publish it to an external system or database.
2. Make sure that the maximum volume of the data you are pulling from the HDFS is medium.

Write MapReduce applications If you want to process large files of data that are on HDFS.

*Using the Hadoop **File Input** step to withdraw the data and process it in PDI is not recommended.*

## *Extending MapReduce Concepts with Pentaho Implementation Architecture*

Pentaho's ETL core engine is distributed and used in the implementation of Hadoop MapReduce functionality. This gives end-users the ability to use Pentaho's tools while running a mapper, combiner, or reducer transformation. The Hadoop framework will manage data locality, node assignment, and `READ` input and `WRITE` output files. Pentaho's engine can then collect the information from the `READ` input file. The information will be delivered to the Hadoop writer using any of the existing steps while running on a mapper or reducer.

There are no limitations or rules that can block a mapper and reducer. Users and developers can add full big file reads, database connections to external sources, and calls to WebServices.

Keep these best practices in mind when working with MapReduce:

- Create a process with mapper only.
    - You can write a mapper-only transformation that can read an input file, do row filtering and transformations of lookups, and write the result by setting the reducer to zero.
    - Keep in mind that there can be multiple mappers reading parts of the input file; therefore, actions like sorting, or any task that suggests you have the full file, is not a good approach.
- Mapper transformations are required to have a Hadoop input and a Hadoop output, but keep these things in mind:
    - Input can be a list of actions to be performed, like WebService calls and emails. Output can be empty.
    - Input can be empty (0 rows or 1 record) and the output can produce millions of rows, whereas real input can be anything from self-generated to external database table.[1]

## *Recommendations for Database Connections*

Do not connect to traditional databases in your mappers and reducers. You can have thousands of mappers or reducers running in parallel, but having all of them hit your database simultaneously can cause an overload of the database and network. There are a limited number of connections that it can handle. Making database connections in a mapper that is instantiated thousands of times will exceed the maximum number of connections that it can handle.

Use alternative technologies, such as distribute key-value stores (HBase, MongoDB, Cassandra), if you need to perform a lookup as part of your MapReduce.

[1] Set Number of mappers to (1) and reducer to (0) and use `org.apache.hadoop.mapred.lib.NLineInputFormat` as `InputFormat`. This reads line per line and does not use data locality for Mapper executing node assignment.

## *Processing Big XML Files*

XML files have a few challenges in the Big Data world. However, there are multiple solutions in place to handle them. Please see Processing XML and JSON in Hadoop using PDI for possible solutions regarding Big Data.

The Apache Mahout project introduces an input format that can be included in Pentaho's libraries for use in MapReduce implementations. For more information on topics discussed here, visit the following links:

- Using a Custom Input or Output Format in Pentaho MapReduce.
- XML Input Format

The Mahout approach:

- XML files cannot be split, and they are not suitable for MapReduce `TextInputFormat` format. Apache's Mahout Project has two classes capable of processing XML files as input formats for MapReduce. The one we use is as follows:

  ```
  org.apache.mahout.classifier.bayes.XmlInputFormat.
  ```

  This `InputFormat` needs to be specified with an open and close tag defined as `xmlinput.start` and `xmlinput.end` in the Pentaho MapReduce **User Defined** section.

- `XmlInputFormat` looks for complete XML sections with `START and END` tags. `XMLInputFormat` will skip records if the file is split, until it reaches a new `START` tag.
- The result of this `XmlInputFormat` is a `Key/Value => Key: file offset` and `Value: XML` section. That is now easy to process with the Pentaho Get data from XML step.

## *Best Practices for Joins and Data*

Avoid doing database lookups or PDI joins inside a mapper or a reducer. We recommend having both datasets you want to join reside in the HDFS when doing lookups for data that resides in a Hadoop cluster. You have several options when using PDI and Hadoop cluster technology to join the datasets. The solution depends on the amount of data that is in the datasets:

*The data row volumes are general guidelines, and performance is also affected by the row sizes.*

### *PDI's Hadoop File Input with Stream Value Lookup Step*

- **Rationale**: This solution is best when the lookup data is less than 10KBs of rows.
- **Solution**: Put the lookup file in Hadoop's distributed cache.

### *PDI's HBase Input Step with Stream Value Lookup*

- **Rationale**: This solution is best when the lookup data is less than 100KBs of rows.
- **Solution**: This will perform full-range scans of the HBase tables. The scans will be slow if those tables are large. HBase configuration will also impact performance.

#### *Directly use HBase API with PDI's User-Defined Java Class*

- **Rationale**: This solution is best when the lookup data has 1MB of rows, or less, and is in HBase tables. It is also best if you are performing single item lookups against large HBase tables.
- **Solution**: You will need to use the HBase Java API in the Pentaho User Defined Java Class (UDJC). You must be familiar with the HBase API features to get the best performance. Use `HBasePools` as a static variable, and try to apply as many filters in a single request to minimize the outgoing calls to HBase. HBase configuration will also impact performance.

#### *MapReduce Joins*

- **Rationale**: This solution is best when both the number of input data rows and the total number of lookup data is large.
- **Solution**: Divide the issue evenly, and ensure the join is done at HIVE level, storing the result in a staging location. The rest of the processing, or other lookups, should be done in MapReduce. Do not put the logic in the SQL `join`. This is normally a big maintenance point if too much logic is added.

For more information about MapReduce joins, please see the following links:

- Joins with MapReduce
- Processing XML and JSON in Hadoop using PDI

### *Best Practices for Writing to Multiple File Destinations*

PDI does not support Hadoop's native feature which provides an output formatter that allows you to write MapReduce output to multiple destinations.

#### *Writing Output Data to Multiple Destinations within a MapReduce*

- **Rationale**: This solution is best for writing multiple mappers or reducers to create multiple files in HDFS. This is probably the easiest way, but would be slowest because you would need to go through the entire dataset multiple times.
- **Solution**: Within the mapper or reducer, create multiple files in HDFS and write data out to these files as needed using Pentaho Hadoop Output step. Each instance of mapper or reducer must make sure that the filename you create in HDFS is unique in the HDFS name space.

### *Best Practices for Compression*

You should enable compression for various stages of MapReduce because Hadoop applications handle large datasets. Hadoop applications will gain the following benefits by using compression:

- **MapReduce is disk IO-intensive**: Using compression significantly reduces the amount of data that is stored in HDFS and intermediate processing files.
- **MapReduce is network-intensive**: You can improve performance by reducing the amount of data that needs to be replicated by HDFS, and by minimizing the number of intermediate files sent over the network during MapReduce.

### *Suggested Compression Algorithms*

- **Rationale**: This solution is best because Snappy, alone, does not support splitting, and will require help when compressing large input files. Use Snappy compression codecs with container file formats such as Sequence File, RCFile, or AVRO, that support splitting and compression.
- **Solution:** Make sure you use a compression codec that supports splitting if input files are compressed and do not use a container file format. Hadoop also supports indexed LZO which is relatively fast and supports splitting. However, you will need to separately install the appropriate libraries on your Hadoop cluster because the indexed LZO implementation is GPL-licensed. Visit Devops Blog for information about compiling and installing Hadoop-LZO compression support module.

- **Rationale**: This solution is best If the compression algorithm you choose for your input files does not support splitting,
- **Solution**: Pre-process the file by chunking it into smaller sizes, preferably <= to HDFS block size. Otherwise, you will effectively get a single mapper to process the entire file. Store the files uncompressed.

The following Hadoop parameters only compress the intermediate mapper output:

- `mapred.compress.map.output`
- `mapred.map.output.compression.codec`

The final output of a MapReduce job is controlled by the following Hadoop parameters:

- `mapred.output.compress`
- `mapred.output.compression.codec`
- `mapred.output.compression.type`

### *Snappy Compression Setup*

CDH4+ and many of the newer distributions ship Snappy compression as part of their distribution. Hadoop can be configured to use Snappy compression for MapReduce output. It is a good idea to have intermediate files compressed using Snappy. This will have the following benefit:

- Reduced temporary storage requirements
- Significantly reduced network traffic during the shuffle or sort phase of MapReduce processing
- A good balance of compression, decompression, and speed when compared to other compression technologies.

The following parameters should be set before using Snappy to compress intermediate files generated by the mapper. In PDI, they are set in the **User Defined** tab of the Pentaho MapReduce step:

- `mapred.compress.map.output = true`
- `mapred.map.output.compression.codec = org.apache.hadoop.io.compress.SnappyCodec`

The following parameters should also be set in order to compress the final output of MapReduce using Snappy compression. In PDI, they are set in the **User Defined** tab of the Pentaho MapReduce step:

- `mapred.output.compress = true`
- `mapred.output.compression.codec = org.apache.hadoop.io.compress.SnappyCodec`
- `mapred.output.compression.type = block`

To show these options in Spoon, various input and output steps from the Snappy libraries must be installed on the client machine and have Spoon include them as part of its configuration. Please see Extracting Data from Snappy Compressed Files for steps on how to do this.

## *Custom JARs*

In addition to PDI's dependent JARs, many PDI applications require third-party Java libraries to perform tasks within PDI jobs and transformations. These libraries must be included in the class path of Hadoop mappers and reducers so PDI applications can use them in the Hadoop cluster. One solution for this is to use the features detailed in the `pmr.kettle.additional.plugins` configuration found on Pentaho's MapReduce webpage.

However, many companies lock down the installs of Pentaho software and do not give users the ability to edit or add files in Pentaho solutions. They have multiple people writing applications, and each application has its own sets and versions of libraries. In this type of environment, the best way to add custom libs is to copy all dependent JARs to Hadoop's Distributed Cache and add the following parameters to the Pentaho MapReduce job step found in the **User Defined** tab:

- `mapred.cache.files`
- `mapred.job.classpath.files`

The following list are some examples of what can be implemented with custom JARs:

- Custom `input` or `Output` format in Pentaho MapReduce
- Custom Practitioner in Pentaho MapReduce

### *Pentaho Application Monitoring*

Pentaho provides real-time performance statistics by using the output step metrics. This can be used to extract step-level statistics.

*You will only get statistics about specific mappers, reducers, or combiners if you use a transformation that works within a Hadoop mapper, reducer, or combiner in the Pentaho MapReduce job step.*

You will not get statistics that are aggregated for all the mapper, combiner, and reducer instances. You will need to have a mapper or reducer store instance data to an external data store, such as HBase or Database. Aggregate this data to get metrics for the entire Hadoop job.

## PDI Cluster on YARN

The PDI YARN Cluster uses the YARN resource manager assignment and resource isolation functions to start multiple CARTE servers inside a Hadoop cluster. It is important to understand that once the CARTE servers are running, the behavior and use cases are the same as any other manually deployed KETTLE CARTE cluster. Furthermore, CARTE servers can `READ` and `WRITE` from within or outside of the Hadoop cluster to from or to files, relational databases, and others. Pentaho's Carte user documentation has more helpful information, as well as documentation on using carte clusters.

The following is a list of YARN Cluster's best use cases:

- Data Ingestion processes
- Real-time on-going transformation
- Assignment of full job and transformation to be executed in one CARTE server
- Use of Hadoop YARN as an escalation platform for PDI transformations when data is not stored in Hadoop

*We recommend that you use Hadoop cluster data locality-aware processes, like MapReduce, for all other use cases where Hadoop data needs to be processed.*

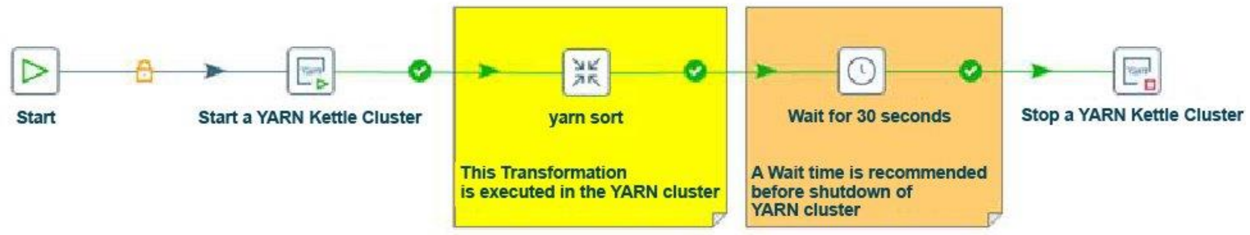
Start
Start a YARN Kettle Cluster
yarn sort
This Transformation is executed in the YARN cluster
Wait for 30 seconds
A Wait time is recommended before shutdown of YARN cluster
Stop a YARN Kettle Cluster

*Figure 3: YARN Transformation Process*

### Best Practice for PDI Cluster on YARN

**User Impersonation Limitation:**

Pentaho YARN cluster currently runs inside the YARN container and uses SHIM configuration to generate a Kerberos ticket. Therefore, it cannot run as a calling user. Visit the Kerberos authentication section in Big Data Integrated Authentication for more information.

# Tips for Writing Better PDI Jobs and Transformations

This section contains information and best practices for creating better PDI Jobs and transformations. Included are steps for handling variables, parameters, and other contents.

## *General PDI Best Practices*

Sub-transformations are not supported in transformations that are used as mappers, reducers, or combiners.

In addition to the best practices detailed in the sections above, the following best practices should be noted:

*Pentaho does not have native support for performing MapReduce directly on MongoDB in a MongoDB cluster.*

PDI offers three steps to write your own processing code:

1. JavaScript
2. Java Expression
3. UDJC

We recommend that all processing that must be done for each row should use the UDJC instead of JavaScript or Java Expression, for best performance. UDJC can perform up to ten times faster than JavaScript.

## *PDI Arguments, Variables, and Parameters*

Do not use hardcode variable and environment settings, such as file paths and Hadoop cluster configuration. Use only variables for those settings. This will allow you to quickly move from different environments without having to change the KJB or KTR. Variables in the Pentaho Documentation has more information on this.

We recommend that you store variables in the following locations:

*Table 2: Variable Locations*

| Variable | Location |
|---|---|
| **Global Variable** | Store in `kettle.properties` and do not change between runs. Typically defines your execution environment through various variables such as development, QA, and production. |
| **Transformation Variable** | Right-click on the canvas of a job or transformation and select **Job** or **Transformation** settings. The scope of a variable is at job or transformation level. These can be set either at command line or at Spoon execute GUI. They can also be passed from jobs to transformations. |
| **Application Specific Variable** | Place the variable in a configuration file, such as `config.properties` and read them in as the first step of the job using the `SET VARIABLE` entry. |

Variables can be passed from a Pentaho job to a transformation that functions as a mapper or reducer. However, the following limitations should be noted:

- Variables cannot be set in a mapper or a reducer and be passed back to parent jobs.
- Variables cannot be set in a mapper and passed to a reducer.

### *Production Environment Performance*

This section provides information on performance for production environments:

- **Rationale**: This solution is best for testing your application with similar sized datasets that you expect in your production environment.
- **Solution**: Use data such as raw input data, lookup and join data, and output data.

- **Rationale**: Performance is dependent on Hadoop technology configurations.
  **Solution**: You must tune the Hadoop cluster, and any related technologies, to fit your load and application scenarios.

- **Rationale**: Row-level logging can degrade performance. Do not perform row-level logging in production environments that use large datasets.
- **Solution**: Do not perform row-level logging in production environments that use large datasets.

- **Rationale:** This solution is best if you are expecting some steps to take a long time to complete.
- **Solution**: You can parallelize them by starting multiple instances of the step. Make sure you do not start multiple instances of steps that write to the same resource, such as file input-output. This type of parallelizing may affect resources that are being parallelized, such as disk, CPU, and memory. Therefore, you must be aware of the extra resources that will be used when parallelizing a step. Adjust resource requirements and step parallelization accordingly.

## *Working with Heavy Loads*

This section offers information for working with heavy loads:

- **Rationale**: Pentaho engineering does not test its products with the non-supported NFS technology.
- **Solution**:  Do not install Pentaho products on an NFS mount. It will work for small loads, but will perform improperly under heavy loads. Pentaho engineering does not test its products with the non-supported NFS technology.

- **Rationale**: Communications between the Carte server and the repository is overactive and can cause performance issues under heavy loads. We recommend you use a file base repository for this type of load.
- **Solution**: Do not use the Pentaho repository if you are using a CARTE cluster and you have thousands of KTR or KJB and you will be running more than 100 jobs and transformations simultaneously.

- **Rationale**: This solution works best if the size of your rows is more than 100KBs.
- **Solution**: You should decrease the number of simultaneous rows that your transformation processes, or increase the JVM heap space and tune the JVM GC.

- **Rationale**: Pentaho products will load all files in directories that it reads for JARs.
- **Solution**: Do not only rename the old JARs when replacing them in a Pentaho directory. You must delete or move them outside of any directory, used by Pentaho, to load classes.

## *Using Mappers and Combiners*

This section covers tips for mappers and combiners:

- **Rationale**: Do not assume that all values will flow through the same mapper.
- **Solution**: The mapper will receive a random set of values.

- **Rationale**: Hadoop decides when to run combiners.
- **Solution**: Do not expect a combiner to always run. Combiners may or may not run, even if you specify one.

## *Working with Multiple Jobs and Large Datasets*

This section provides information for working with jobs and datasets:

- **Rational**: You may need more space, depending on how many simultaneous jobs you are running, if you are using a Carte cluster that uses repository.
- **Solution**: For Linux systems, increase the limit of the system and user that PDI applications run to around 16K.

### *Using Reducers*

This section offers information about using reducers:

- **Rationale**: Reducer logic should presume operations for one key at a time.
- **Solution**: Code the reducer. All the values for the same key will flow through the same reducer.

- **Rationale**: A reducer gets all data associated with a single key.
- **Solution**: Do not expect a reducer to sum or count values across keys.

## *MapReduce JVM Settings*

Hadoop will create a new JVM for each instance of a mapper and a reducer, by default. You can set various JVM options, such as `-Xms` and `–Xmx`, using the following Hadoop parameters:

*Table 3: Hadoop Parameters*

| Parameter | Defintion |
|---|---|
| **`mapred.child.java.opts`** | Applies to all task JVMs, such as mappers and reducers. |
| **`mapred.child.java.opts`** | If present, it is used over `mapred.child.java.opts` for mapper. |
| **`mapred.reduce.child.java.opts`** | If present, it is used over `mapred.child.java.opts` for reducer. |

These parameters can be set cluster wide in `mapred-site.xml`. In addition, if you want to override the system settings on a per job basis, you can also set them in PMR under the **User Defined** tab.

## *PDI Operations Mart*

Do not collect the operational data using PDI Operations Mart (POM) for transformations that are used as mappers or reducers, and Hadoop jobs that have more than a few mapper or reducer instances. When transformations are configured to use POM, they will open a connection to the its database to add a record for collected statistics. When this is done in a Hadoop cluster, you could have thousands of mappers or reducers opening connections to the database and overwhelming it with too many connections. There are alternate steps to collect statistics:

### *Collecting Statistics with the Output Metric Step*

- **Rationale**: This solution is best for generating a row of statistics every second. You can collect statistics in the transformation using the output metrics step.
- **Solution**: Write the output of the output steps metrics to a file in HDFS. Be sure that the file name is unique for each mapper and reducer instance. Alternatively, you can put the data into HBase, MongoDB or Cassandra. Visit Output Step Metrics to view the few issues associated. Write a Pentaho MapReduce application that will aggregate the data and publish the statistics you want. Alternatively, take the data and populate the POM database.

## Related Information

Please visit the following links for more information about topics discussed here:

**Pentaho Best Practices:**

- Pentaho BA Server and High Availability

**Pentaho Documentation**:

- Using Carte Clusters
- Variables
- Output Step Metrics
- Pentaho Big Data Integrated Authentication.

**Pentaho Wiki**:

- Get data from XML
- Extracting Data from Snappy Compressed Files
- MapReduce
- Carte User Documentation
- Create Mapper and Reducer for Aggregate Dataset
- Using a Custom Input or Output Format in Pentaho MapReduce
- Create Mapper Transformation to Parse Weblog File

**Tech Ramblings Blog**:

- Processing XML and JSON in Hadoop using PDI
- Pentaho MapReduce Workflow

**Hadoop Wiki**:

- Partitioning Your Job into Maps and Reduces

**Apache Mahout**:

- XML Input Format
- Apache Mahout project

**General Blogs**

- Source Open – Joins with MapReduce
- Devops Blog – Compiling and Installing Hadoop-LZO Compression Support Module

## Best Practice Check List

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 |
|---|---|---|
| **Is Pentaho Map Reduce used in the Project?** | YES ___ NO ___ | |
| **Are external or customized libraries required for Map Reduce?** | YES ___ NO ___ | |
| **Is it required to do Joins of Big Data sets? What method was selected?** | YES ___ NO ___ | |
| **Is HBase or Cassandra used as part of mapper or reducer?** | YES ___ NO ___ | |
| **Is PDI Cluster on YARN used in the project?** | YES ___ NO ___ | |
| **Is Avro format used?** | YES ___ NO ___ | |
| **Is Parquet format used?** | YES ___ NO ___ | |
| **Is Compression used?** | YES ___ NO ___ | |