Big Data On-Cluster Processing

with Pentaho MapReduce

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

This page intentionally left blank.

Overview

This document covers some best practices to push exchange, transform, and load (ETL) processes to Hadoop-based implementations. When working on big data implementations, we must consider concepts like data locality, distributed processing, resource assignment isolation, and processing queue.

The Pentaho Data Integration (PDI) tool includes multiple functions to push work to be done on the cluster using distributed processing and data locality acknowledgement. This document covers use cases and situations where different methods are recommended.

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.x

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

Pentaho Deployment Architecture Recommendations

The Pentaho Server needs to communicate with the Hadoop Distributed Filesystem (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.
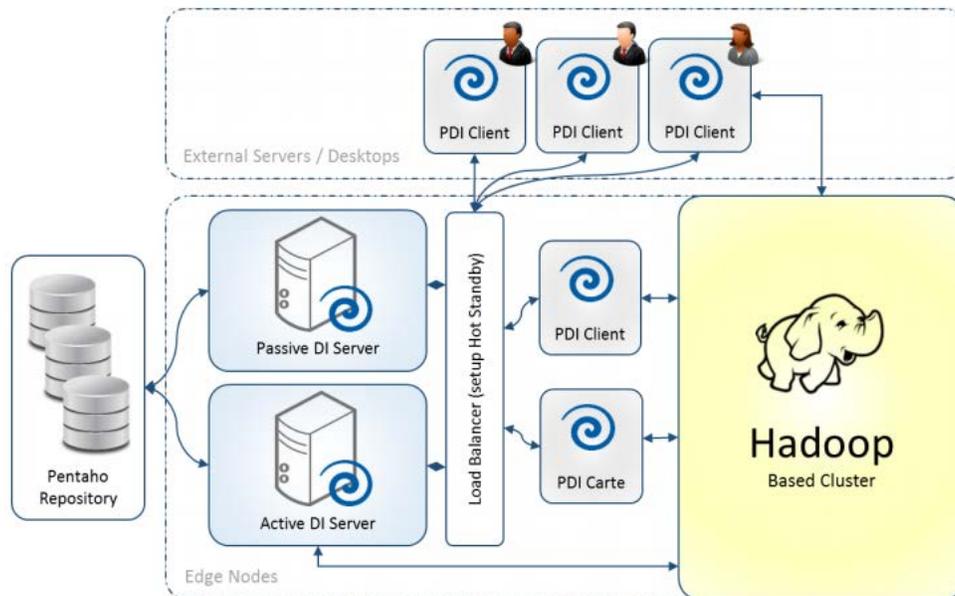

Figure 1: PDI Server Deployment

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 on these topics, see *Pentaho Tools and Integrated Authentication Methods* in the Security for Pentaho library.

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 here:

Table 1: `plugin.properties` Locations

Target	Plugin.Properties Location
Pentaho Server (Pentaho 7.x and later)	`[PENTAHO_INSTALL_HOME]/server/pentaho-server/pentaho-solutions/system/kettle/plugins/pentaho-big-data-plugin/plugin.properties`
DI Server (Pentaho 6.x and earlier)	`[PENTAHO_INSTALL_HOME]/server/data-integration-server/pentaho-solutions/system/kettle/plugins/pentaho-big-data-plugin/plugin.properties`
BA Server (Pentaho 6.x and earlier)	`[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/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`

Writing to HDFS

We recommend giving the user who executes PDI permission to write to the root HDFS directory when you set up the HDFS. You can manually create the `/opt/pentaho` directory and give the PDI user write permission, and 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.

Running Multiple Pentaho Versions

PDI will allow you to run multiple versions on the same Hadoop cluster. This setup will let you test PDI upgrades.

Delete the `pmr.kettle.dfs.install.dir/[pdi-version-hadoop-version]` directory in HDFS if you change PDI's dependent JARs with 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.

- [Orchestration](#)
- [Pentaho MapReduce](#)
- [Custom JARs](#)
- [PDI Cluster on YARN](#)

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.
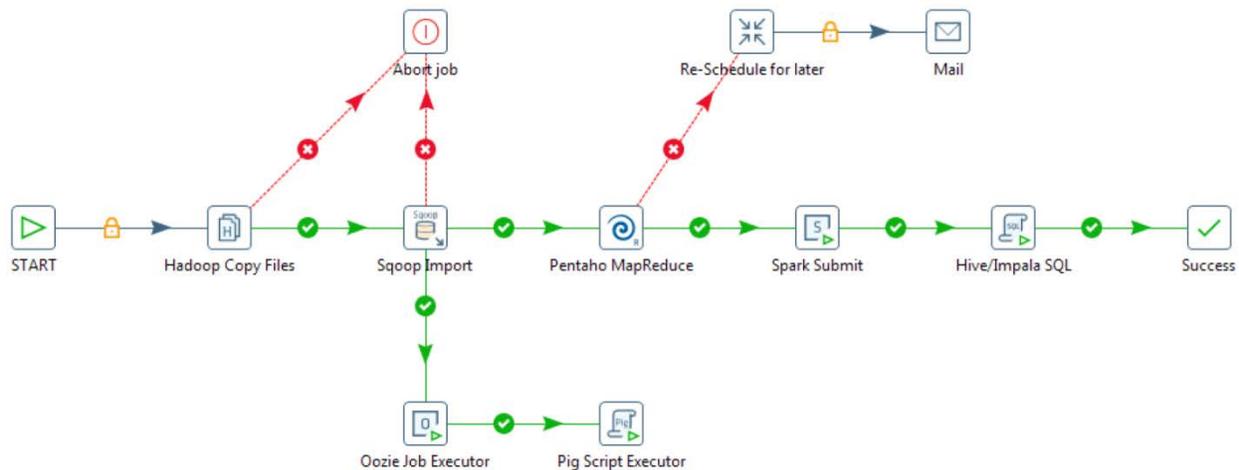
Figure 2: PDI Orchestration

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 transformations** and **reducer transformations**.
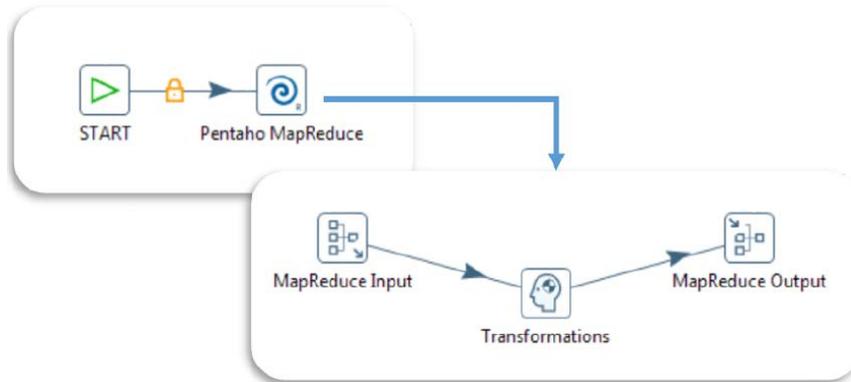
Figure 3: File Parsing

Find standard and common usage of the MapReduce functionality at the following:

- [Pentaho MapReduce workflow](#)
- [Create Mapper and Reducer for Aggregate Designer](#)
- [Create Mapper Transformation to Parse Weblog File](#)

Best Practices for MapReduce

Our recommendations for MapReduce include the following:

For Pentaho MapReduce job steps:

1. 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.

For 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 WebServices 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 tables.[1]

Recommendations for Database Connections

Do not connect to traditional databases in your mappers and reducers. You can have thousands of mappers and 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 distributed 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 by line and does not use data locality for the mapper executing node assignments.

Processing Big XML Files

XML files have a few challenges in the big data world. However, there are several solutions, including those in *Parsing XML on PDI* in the [Big Data and Pentaho](#) library, in place to handle them.

The [Apache Mahout project](#) introducers an input format that can be included in Pentaho's libraries for use in MapReduce implementations. More information can be found at [Using a Custom Input or Output Format in Pentaho MapReduce](#) and in the XML Input Format section of *Parsing XML on PDI* in the [Big Data and Pentaho](#) library.

In the Mahout approach:

- XML files cannot be split, and they are not suitable for the 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.

Table 2: Joins and Data

Method	Details
PDI's Hadoop File Input with Stream Value Lookup Step	This solution is best when the lookup data is less than 10KBs of rows. Put the lookup file in Hadoop's distributed cache.
PDI's HBase Input Step with Stream Value Lookup Step	This solution is best when the lookup data is less than 100KBs of rows. 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.

Method	Details
Directly use HBase API with PDI's User-Defined Java Class	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. 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 as possible in a single request to minimize the outgoing calls to HBase. HBase configuration will also impact performance.
MapReduce Joins	This solution is best when both the number of input data rows and the total amount of lookup data is large. 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 [Joins with MapReduce](#) and *Parsing XML on PDI* from the [Big Data and Pentaho](#) library.

Best Practices for Writing to Multiple File Destinations

PDI does not support Hadoop's native feature that provides an output formatter that allows you to write MapReduce output to multiple destinations. Instead, write output data to multiple destinations within a MapReduce. 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.

To do this, within the mapper or reducer, create multiple files in HDFS and write data out to these files as needed using the **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 namespace.

Best Practices for Compression

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

- **MapReduce is disk I/O 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.

Table 3: Suggested Compression Methods

Method	Details
Use Snappy Compression Codecs with Container File Formats	Because Snappy alone does not support splitting, and requires help when compressing large input files, try using Snappy compression codecs with container file formats such as Sequence File, RCFile, or Avro, that support splitting and compression. Hadoop also supports indexed LZO which is relatively fast and supports splitting, but 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.
Preprocess File by Chunking	Preprocess 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. This solution is best if the compression algorithm you choose for your input files does not support splitting.

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 benefits:

- 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 classpath 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 at [Pentaho MapReduce](#).

However, many companies lock down the installations 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 libraries is to copy all dependent JARs to Hadoop's distributed cache and add the following parameters to the **Pentaho MapReduce** job step's **User Defined** tab:

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

The following are 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 can 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. For that, you will need 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. Once the Carte servers are running, the behavior and use cases are the same as for any other manually deployed Kettle Carte cluster.

Furthermore, Carte servers can read and write from within or outside of the Hadoop cluster, to or from files, relational databases, and others. Pentaho's Carte user documentation has more helpful information, as well as documentation on using Carte clusters.

The YARN cluster is most useful in cases such as:

- Data ingestion processes
- Real-time ongoing 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 you use Hadoop cluster data locality-aware processes, like MapReduce, for all other use cases where Hadoop data needs to be processed.
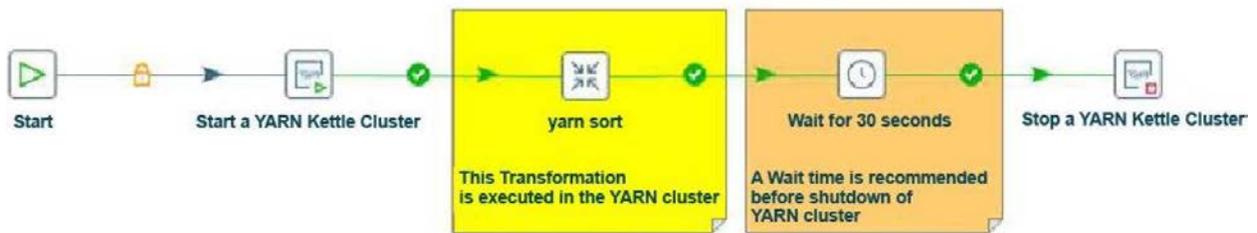

Figure 4: YARN Transformation Process

Write Better PDI Jobs and Transformations

This section contains information and best practices for creating better PDI jobs and transformations.

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

- General PDI Best Practices
- PDI Arguments, Variables, and Parameters
- Production Environment Performance
- Working with Heavy Loads
- Using Mappers and Combiners
- Working with Multiple Jobs and Large Datasets
- Using Reducers
- MapReduce JVM Settings
- Operations Mart

General PDI Best Practices

Note that:

- Subtransformations are not supported in transformations that are used as mappers, reducers, or combiners.
- 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. **Modified JavaScript value**
2. **User defined Java expression**
3. **User defined Java class (UDJC)**

We recommend that all processing that must be done for each row use **UDJC** instead of **Modified JavaScript value** or **User defined Java expression**, for best performance. **UDJC** can perform up to ten times faster than **Modified JavaScript value**.

PDI Arguments, Variables, and Parameters

Do not use hardcoded variables and environment settings, such as filepaths and Hadoop cluster configuration. Use only variables for those settings. This will allow you to quickly move among different environments without having to change the job or transformation. Variables in the Pentaho Documentation has more information on this.

We recommend that you store variables in the following locations:

Table 4: Variable Storage Locations

Variable	Location
Global Variable	Store in `kettle.properties` and do not change between runs. Typically defines your execution environment such as development, QA, and production, where you will use the same variables, but with different values, for each environment.
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 the 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 it 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, please note the following limitations:

- 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:

Table 5: Production Environment Performance Improvements

Recommendation	Details
Use data such as raw input data, lookup and join data, and output data.	This solution is best for testing your application with similar-sized datasets that you expect in your production environment.
Tune the Hadoop cluster, and any related technologies, to fit your load and application scenarios.	Performance is depending on your Hadoop technology configurations.
Do not perform row-level logging in production environments that use large datasets.	Row-level logging can degrade performance.
Parallelize steps by starting multiple instances of them.	If you are expecting some steps to take a long time to complete, try parallelizing. Do not start multiple instances of steps that write to the same resource, such as file input/output. Be aware of extra resources (disk, CPU, memory, and so on) used when parallelizing as this may affect them, and adjust accordingly.

Working with Heavy Loads

This section offers information for working with heavy loads:

Table 6: Recommendations for Heavy Workloads

Recommendation	Details
Do not install Pentaho products on an NFS mount.	Installation on an NFS mount 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.
Do not use the Pentaho repository if you are using a Carte cluster and have thousands of transformations and jobs and will be running more than 100 jobs and transformations simultaneously.	Communication between the Carte server and the repository is overactive and can cause performance issues under heavy loads. We recommend you use a file-based repository instead for this type of load.
Decrease the number of simultaneous rows your transformation processes, or increase the JVM heap space and tune JVM garbage collection.	This solution works best if the size of your rows is more than 100 kilobytes.
Old JARs you are replacing must be deleted or moved outside of any directory used by Pentaho.	Since Pentaho products will load all files in directories that it reads for JARs, if you do not delete old JARs or move them outside of any directory Pentaho uses, you will not be able to load classes properly.

Using Mappers and Combiners

This section covers tips for mappers and combiners:

- The mapper will receive a random set of values. Do not assume that all values will flow through the same mapper.
- Hadoop decides when to run combiners, so do not expect that a combiner will always run. Combiners may or may not run, even if you specify one.

Working with Multiple Jobs and Large Datasets

You may need more space, depending on how many simultaneous jobs you are running, if you are using a Carte cluster that connects to an Enterprise Edition (EE) repository. For Linux systems, increase the open files limit for the user who executes PDI to around 16K.

Using Reducers

When using reducers:

- Code the reducer. All the values for the same key will flow through the same reducer. Reducer logic should presume operations for one key at a time.
- A reducer gets all data associated with a single key. 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 7: Hadoop Parameters

Parameter	Details
`mapred.child.java.opts`	Applies to all task JVMs, such as mappers and reducers.
`mapred.map.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.

Operations Mart

Do not collect the operational data using Pentaho Operations Mart (Ops Mart) 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 Ops Mart, they will open a connection to the 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.

To collect statistics, you can try this alternative:

- Write the output of the output steps metrics to a file in HDFS. Be sure that the filename is unique for each mapper and reducer instance.
- Alternatively, you can put the data into HBase, MongoDB, or Cassandra.
- Write a Pentaho MapReduce application that will aggregate the data and publish the statistics you want.
- Alternatively, take the data and populate the Ops Mart database.

Related Information

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

- Pentaho
  - [Big Data and Pentaho Library](#)
  - [Carte User Documentation](#)
  - [Components Reference](#)
  - [Create Mapper and Reducer for Aggregate Designer](#)
  - [Create Mapper Transformation to Parse Weblog File](#)
  - [Get Data from XML Step](#)
  - [Pentaho MapReduce](#)
  - [Pentaho MapReduce Workflow](#)
  - [Security for Pentaho Library](#)
  - [Using Carte Clusters](#)
  - [Using a Custom Input or Output Format in Pentaho MapReduce](#)
  - [Variables](#)
- Apache
  - [Apache Mahout Project](#)
  - [Hadoop Wiki](#)
- Blogs
  - [Compiling and Installing Hadoop-LZO Compression Support Module](#)
  - [Joins with 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
Is Pentaho MapReduce used in the project?	YES_____ NO_____	
Are external or customized libraries required for MapReduce?	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_____	