

A large, decorative blue spiral graphic is positioned on the left side of the dark blue banner, partially overlapping the white background.

Best Practices - PDI Performance Tuning

This page intentionally left blank.

Contents

- Overview 1
- Performance Tuning Process..... 1
- Identifying, Eliminating, and Verifying Bottlenecks2
 - Identifying Bottlenecks with Step Metrics2
 - Eliminating Bottlenecks3
 - Verifying Bottlenecks4
- External Performance Factors5
 - Network Performance5
 - Data Source and Target Performance.....6
 - Storage Performance.....7
- PDI Performance Tuning9
 - PDI Transformation Design9
 - Query Management.....9
 - Database Management.....10
 - Real-time Data on Demand10
 - Scripting.....11
 - PDI Job Design12
 - Looping.....12
 - Scaling Up Transformations13
 - CPU and Multithreading.....13
 - Memory Utilization13
 - PDI Clusters.....14
 - Clustered Transformations.....15
 - Partitioning16
 - PDI Visual MapReduce for Hadoop18
- Related Information20

This page intentionally left blank.

Overview

This guide provides an overview of factors that can affect the performance of Pentaho Data Integration (PDI) jobs and transformations, and provides a methodical approach to identifying and addressing bottlenecks.

Some of the things discussed here include identifying, eliminating, and verifying bottlenecks, external performance factors, and PDI performance tuning.

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

Software	Version
Pentaho	5.4, 6.x, 7.x

Performance Tuning Process

PDI transformations are Extract, Transform, and Load (ETL) workflows that consist of steps linked together as shown below:

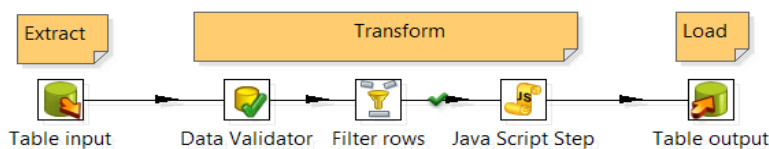


Figure 1: ETL Workflow

There are three basic types of steps:

- **Input step** – Ingests data into PDI (e.g. **Table input** step).
- **Transformation step** – Processes data within PDI (e.g. **Data Validator**, **Filter rows**, and **Java Script** steps).
- **Output step** – Outputs transformed data from PDI (e.g. **Table output** step).

All steps are processed in parallel, and the overall speed of the transformation is capped at the speed of the slowest step. Therefore, the following process is used to improve transformation performance:

1. Identify the slowest step (the bottleneck).
2. Improve performance of the slowest step until it is no longer the bottleneck.
3. Repeat steps 1 and 2 for the new bottleneck. Iterate until the Service Level Agreement (SLA) is met.

Identifying, Eliminating, and Verifying Bottlenecks

Several PDI features provide techniques for working with bottlenecks, including:

- [Identifying Bottlenecks with Step Metrics](#)
- [Eliminating Bottlenecks](#)
- [Verifying Bottlenecks](#)

Identifying Bottlenecks with Step Metrics

Row buffers are created between each step. This allows the steps to retrieve rows of data from their inbound row buffer. The rows are then processed and passed to an outbound row buffer that feeds into the subsequent step. Row buffers can hold up to 10,000 rows, but this can be configured for each transformation.

The **Step Metrics** tab on the **Execution Results** pane will show real-time statistics for each step when you run a transformation. The **Input/Output** field shows a real-time display of the number of rows in the buffers feeding into and coming out of each step. You know that the step cannot keep up with the rows being fed into it if the input of a step is full. Below are some examples.

Example of Table Input Bottleneck

The following example shows a snapshot in real-time of a running transformation:

Table 1: Table Input Bottleneck

Step Name	Input/Output
Table Input	0/50
Data Validator	48/52
Filter Rows	54/42
JavaScript Step	37/51
Table Output	43/0

The transformation has no trouble keeping up with incoming rows because the buffers are low (much less than 10,000). This allows them to be processed and delivered to the target. Therefore, the **Table Input** step is the bottleneck.

Example of Table Output Bottleneck

The following example shows that the buffers are full (close to the buffer size of 10,000):

Table 2: Table Output Bottleneck

Step Name	Input/Output
Table Input	0/9720
Data Validator	9850/9741
Filter Rows	9922/9413
JavaScript Step	9212/9413
Table Output	9985/0

PDI is waiting for the **Table Output** step to consume rows. This means that the data target is the bottleneck.

Example of Transformation Bottleneck

The following example shows that the row buffers are filled all the way through to the **JavaScript** step:

Table 3: Transformation Bottleneck

Step Name	Input/Output
Table Input	0/9815
Data Validator	9784/9962
Filter Rows	9834/9724
JavaScript Step	9834/27
Table Output	53/0

The **Table Output** buffers are low, which shows that the data target has no trouble consuming output from PDI. This indicates that the **JavaScript** step is the bottleneck.

Eliminating Bottlenecks

Consider the following things to detect and eliminate bottlenecks:

Table 4: Eliminating Bottlenecks

Solution	Explanation
Performance Monitoring	Real-time Performance Monitoring captures throughput in rows-per-second for each step, for several metrics. Performance monitoring values can be stored in a centralized logging database to enable analyzing jobs that are run on remote PDI servers. Performance monitoring requires additional resources and can be enabled or disabled on individual jobs and transformations.
Repeat Measurements	Data caching can significantly affect performance for subsequent runs. For example, a database may cache query results so that, the next time the query is run, it will return results much faster. Make sure to measure the same scenario several times to account for caching.

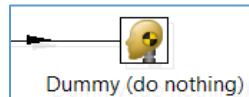
Follow the guidelines in the [External Performance Factors](#) section below if the **Input** or **Output** step is the bottleneck. This will help you address areas such as networking, database, or storage optimization that can affect how quickly data can be imported or exported from PDI. Otherwise, the [PDI Performance Tuning](#) section gives suggestions for improving performance within PDI.



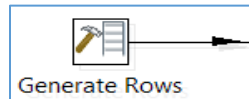
*We recommend selecting the **Metrics** tab on the **Execution Results** pane to view the length of time in milliseconds for initialization and execution of each transformation step. This can assist with identifying bottlenecks.*

Verifying Bottlenecks

You can verify that the bottleneck is an **Output** step by replacing it with a **Dummy** step, which throws away the rows. It is likely that the **Output** step is the bottleneck if the overall speed of the transformation increases.



You can replace an **Input** step with a **Generate Rows** step, **Data Grid** step, or a **Text file input** step that is pointing to a file on a fast, local storage or a Random-Access Memory (RAM) drive. You can then check if the transformation is faster.



Follow guidelines in the [PDI Performance Tuning](#) section if the bottleneck step is a transformation step. This will help you improve PDI performance.

External Performance Factors

External factors, such as network or database performance, are likely the problem if the bottleneck is an **Input** or **Output** step. PDI is part of a larger system that includes data sources, data targets, networking, storage, and other infrastructure components. This section discusses these areas but does not provide detailed tuning instructions, such as how to tune your Oracle database.

Network Performance

Many times, the network is the bottleneck and throughput is capped by the network layer.

First, eliminate the network as the bottleneck by following these steps:

1. Export the source data to a local text file and measure how long it takes for the database to export the data, without touching the network.
2. Copy the text file to the PDI server and measure the time for the transfer.
3. Modify the transformation to import the local file.
4. Run the transformation again and measure the time to import the local file, without touching the network.
5. Compare these measurements to assess network performance.

Consider the following things to assess for possible network bottlenecks:

Table 5: Bottleneck Solutions

Solution	Explanation
Network Sizing	Consider adding additional Network Interface Controllers (NIC) or upgrading to 10gbps. Scale out data sources, targets, and PDI using clustering technology, which leverages network connectivity across multiple servers. Ethernet bonding can provide increased throughput as well as failover.
Network Bottlenecks	Switches, routers, proxy servers, firewalls, and other network appliances can create bottlenecks. Consider upgrading or bypassing these altogether.
WAN Optimization	Moving data across a Wide Area Network (WAN) presents several challenges. Consider moving data sources, data targets, or PDI servers to the same Local-Area Network (LAN). There are several techniques and third-party appliances designed to improve throughput if you must move data across a WAN. One alternative to direct database connections is to dump data to a text file and perform a file transfer using a WAN optimized tool, such as ExpeDat . Please visit WAN optimization for more information.

Solution	Explanation
Cloud Computing	Network configuration in the cloud can cause issues due to the lack of transparency in the implementation. Please visit Amazon EC2 Instance Types and Pentaho and Amazon Web Services for more information about dealing with cloud computing.
Ephemeral Storage	Use a local attached Solid State Drive (SSD) instead of an Elastic Block Store (EBS), which sits on a Network-Attached Storage (NAS). View the Storage Performance section below for data loss considerations.
Offline Shipping	In extreme cases, it is faster to ship hard drives overnight to far off locations, avoiding the network altogether. Large data migration efforts are commonly used for offline shipping. Please visit Sneakernet for more information about offline shipping.

Data Source and Target Performance

The performance of the data source or target can also be the cause of a bottleneck. Database optimization is a technique for managing performance. Some of the more common approaches to database optimization are listed below:

Table 6: Database Optimization Techniques

Technique	Definition
Query Optimization	Many databases provide a <code>SET EXPLAIN</code> feature that allows you to determine whether indexes are being used or if the database is performing a complete table scan. Constraints and triggers can also affect performance.
Local Export/Import	Import or export a local text file or pipe to <code>/dev/null</code> and measure the throughput. This may represent the fastest throughput possible for the data source or target.
Bulk Loaders	Many databases provide bulk loaders that may be faster than performing insert queries. PDI includes bulk loader transformation steps for several databases. PDI also supports calling command-line bulk loaders.
Data Staging/Pre-processing	Consider creating a materialized view, pre-processing data on the database, or loading a staging table. These approaches can simplify the ETL logic and possibly reduce data volume over the network.
Database Technologies	Hadoop, NoSQL, analytical, multi-dimensional, in-memory, cache, and other database technologies can provide better performance for certain situations.

Technique	Definition
Replication	Database replication allows a mirror image of a database to be kept close to PDI. This can reduce or even eliminate network connectivity between PDI and the data source or target.
Database Design	Star schemas and other data mart design patterns can dramatically improve performance at the cost of additional complexity and resources.
Clustering/Sharding/Partitioning	Some databases support table partitioning or database sharding, which can improve the performance of certain types of queries. PDI has functionality for configuring transformations to leverage these features. View the PDI Clusters section below for more information about leveraging these features.

Storage Performance

Data may need to be stored outside of the database when working with data files, staging, batching, archives, and more. Use the following table as a guide for choosing the correct storage option. The throughput (MB/s) shown below are only rough estimates:

Table 7: Storage Performance

Solution	Approx. MB/S	Explanation
RAM Disk	17,000	RAM is the fastest hardware storage option. The operating system (OS) can be configured to cache files in RAM. These drives are easily created in Linux or Unix and mounted to any path like a regular hard drive. Frequently used files can be cached or staged on RAM drives for fast access or processing. RAM is expensive, volatile (erased on reboot), and limited in capacity.
SSD	2,000	SSDs provide fast, non-volatile (permanent) storage mounted as a local hard drive. These can come in the form of a Peripheral Component Interconnect Express (PCIe) card installed on the server motherboard. SSDs also provide fast, random access compared to rotational media.
NAS/SAN	30	NAS and Storage Area Networks (SAN) provide fail-over, redundancy and (optionally) disaster recovery, offsite backup, huge capacity, and more. These typically provide access through the Network File System (NFS), Common Internet File System (CIFS), or the Internet Small Computer Systems Interface (iSCSI). Third party vendors can provide local NAS or SAN storage inside the data centers of cloud providers, such as Amazon Web Services (AWS). This can provide a high-performance alternative to S3 and EBS.

Solution	Approx. MB/S	Explanation
AWS EBS	30-125	EBS is provided by AWS. It is approximately ten times more durable than physical HDDs due to replication on back-end NAS. Snapshots or RAID10 is still recommended. Striping EBS volumes can increase performance and capacity.
AWS Glacier	See description	AWS Glacier is a low-cost, long-term, cold storage. When you make a request to retrieve data from Glacier, you initiate a retrieval job. Once the retrieval job completes, your data will be available to download for 24 hours. Retrieval jobs typically finish within three to five hours. Upload and download speeds may be similar to S3.
AWS S3	1	The eventual consistency model is a major factor affecting usage. S3 performance is also much slower than EBS. Please visit Amazon S3 for more information.
vHDD (virtual HDD)	Depends on type	<p>Virtual hard drives (vHDD) are used by virtual machines (VM). The vHDD is presented to the VM as a local hard drive. These are typically files stored on an NAS or SAN but can be locally attached storage as well.</p> <p>Performance, cost, capacity and other specifications depend on multiple factors. These include cost of the storage server, capacity limits imposed by the file system, or VMware. Some other factors may include cloud infrastructure, speed of networking and storage servers, load on shared resources, and more.</p> <p>vHDDs can be thin-provisioned (i.e. a 100GB vHDD) with 10GB data. It will only occupy 10GB on backend storage, but will appear as 100GB to the VM's OS.</p> <p>vHDDs can also be expanded easier than physical storage. In some cases, the Logical Volume Manager (LVM) can support expansion of a vHDD with zero down time.</p>
HDD (physical)	750	The throughput shown is for a single hard drive. RAID configurations can provide redundancy, fail-over, higher capacity, faster throughput, and lower latency. Rotational media can be significantly slower for random access compared to RAM and SSD.

PDI Performance Tuning

PDI performance is likely the issue when there is a transformation bottleneck. This section provides techniques for tuning various aspects of PDI, including:

- [PDI Transformation Design](#)
- [PDI Job Design](#)
- [Scaling Up Transformations](#)
- [PDI Clusters](#)
- [PDI Visual MapReduce for Hadoop](#)

You should start with optimizing ETL to be as efficient as possible, and then evaluate platform-related factors, such as hardware sizing and clustering.

PDI Transformation Design

PDI contains several techniques for designing and building ETL transformations. This section contains best practices for maximizing transformation performance.

Query Management

The following table discusses techniques for managing queries to improve transformation performance:

Table 8: Query Management

Technique	Definition
Data Caching	High network latency can make executing multiple queries much slower than running a single bulk query. Most lookup steps give you cache lookup values. You can also perform up-front loading of records in a single query and cache the results, instead of performing multiple queries
Batch Updates	Batch updates can also reduce the number of queries. The commit size setting controls the size of the batches. Please visit the Table Output Step for more information about using batch updates.

Database Management

The following table discusses various techniques for managing your database to improve transformation performance:

Table 9: Database Management

Technique	Definition
Database Sorting	Sorting on the database is often faster than sorting externally, especially if there is an index on the sort field(s). Visit the Memory Utilization section below for information on configuring the Sort rows step to leverage memory and CPU resources.
Prepared Statements	Most database steps prepare statements, which incurs some overhead up front but improves performance overall. The Execute SQL Script , Execute row SQL Script , and Dynamic SQL row steps do not perform this initial statement preparation and may not perform as well.
Database Processing	Performance will be better if data is processed directly on the database, in some cases. This approach may eliminate the need for a PDI transformation. Data can be pre-processed or staged on the database to simplify PDI transformations. Transformation logic can also be moved to the target sources in an ELT design pattern. Stored procedures, triggers, materialized views, and aggregation tables are just some of the techniques that can be leveraged.

Real-time Data on Demand

PDI contains various tools for viewing data in real-time, including **Report Bursting**. PDI transformations can feed results into a PDI report template and burst the report out through email, or to a file server without having to stage the data in a reporting table.

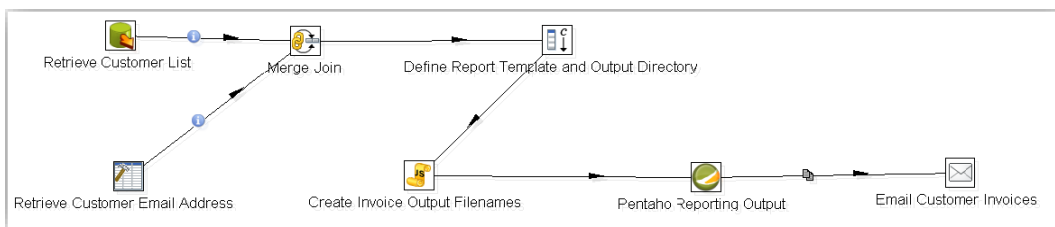


Figure 2: Report Bursting

Some other techniques for viewing data in real-time include:

- **Extract, Transform, and Report**– PDI transformations support Extract, Transform, and Report (ETR). Pentaho reports and dashboard frameworks can use PDI transformations as a native data source.
- **PDI Thin JDBC driver**– Any application that connects to an Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) data source can send an SQL query to a PDI transformation. It does this by using the PDI JDBC driver. PDI will parse the `where` clause and pass criteria into transformation parameters that can drive the logic of the transformation. The transformation feeds the results back to the client application as a normal JDBC query result set. This can support near real-time analytics.

Scripting

The **JavaScript** step provides enormous flexibility, but it may not perform as well as other highly optimized, single-purpose transformation steps. The following table provides techniques for improving **JavaScript** step performance:

Table 10: JavaScript Performance Techniques

Technique	Definition
Compatibility mode	Turn off compatibility mode when not needed. This will run the newer, faster JavaScript engine.
Step plugin	Consider writing a step plugin. This can provide better performance than using a JavaScript step.
User Defined Java Class	A User Defined Java Class step may perform better than a JavaScript step.

Some other things to consider when designing transformations include:

- **Constant and Static Values**– Avoid calculating the same static value on every row. You can simply calculate constants in a separate transformation and set variables to be used in downstream transformations. You can also calculate constants in a separate stream and use the **Join Rows (Cartesian product)** step to join the constant into the main stream.
- **Lazy Conversion**– This setting will postpone data conversion as long as possible. This includes character decoding, data conversion, and trimming. This can be helpful if certain fields are not used, if data will be written out to another text file, or in some bulk loader scenarios.
- **NIO Buffer Size**– This parameter determines the amount of data read at one time from a text file. This can be adjusted to increase throughput.
- **Performance Monitoring and Logging** – Detailed [Performance Monitoring and Logging](#) can be very helpful in development and test environments. The logging level can be turned down and performance monitoring can be disabled for production environments to conserve resources. Please visit [PDI Performance Tuning Tips](#) for more information about logging and performance monitoring.

PDI Job Design

PDI contains several techniques for designing and building ETL Jobs. This section provides best practices for improving job performance.

Looping

Avoid creating loops in PDI jobs. In the example below, the **Get customer info** transformation gets a customer record, and then the **Send report** transformation sends a report to that customer. The **Send report** transformation continues to the **DUMMY** job entry and loops back to **Get customer info**. It retrieves the next customer and the loop continues until there is no data left.

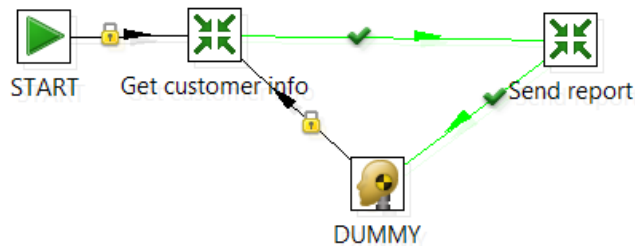


Figure 3: Looping

Set the **Execute for every input row** setting instead of creating a loop in the job.

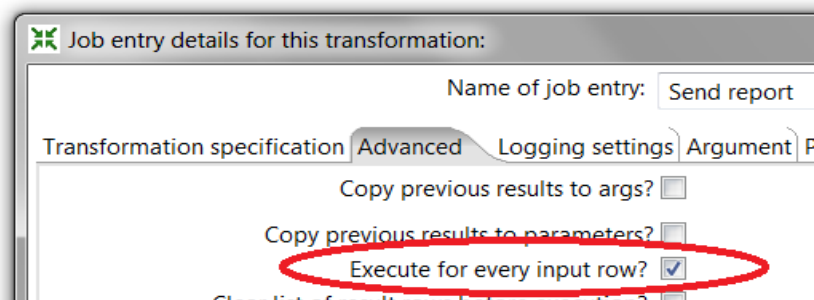


Figure 4: Job Entry Details

The **Get customers info** transformation will retrieve the customers and send them to the **Send report** transformation, which will run once for every incoming row. This approach achieves the same result and will perform better.

Some other techniques to consider when designing ETL jobs include:

- **Database Connection Pooling**– There is some overhead with establishing new database connections at run time. Enable connection pooling to maintain a pool of open connections that can be used as needed by the job or transformation.
- **Checkpoints**– You can specify checkpoints in your jobs and restart jobs from the last successful checkpoint. This avoids having to re-start jobs from the beginning in case of failure.

Scaling Up Transformations

This section describes how transformations and jobs can be configured to leverage memory and CPU resources.

CPU and Multithreading

PDI transformations are multithreaded. This means you can increase the number of copies of a step to increase threads assigned to that step, which allows you to assign more CPU resources to slower steps. Each step in a transformation gets its own thread, and transformation steps run in parallel. Doing this leverages multiple cores, for example:

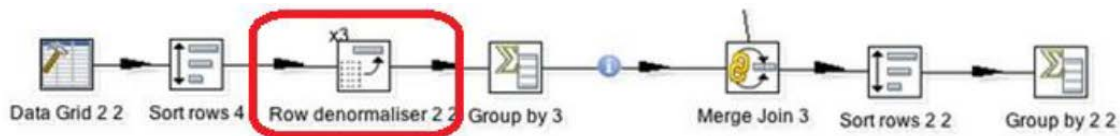



Figure 5: Row Denormaliser

The **Row denormalizer** step is assigned three step copies that will spawn three threads. Each of the other steps will spawn its own thread. Therefore, the transformation will spawn a total of nine threads that could utilize up to nine cores on the PDI server.

 *Make sure not to allocate too many threads, as this can degrade performance. Optimal performance is achieved by keeping the number of steps to less than three to four times the number of cores.*

Some other techniques for leveraging memory and CPU resources include:

- **Blocking Steps**– The **Blocking** step and **Block this step until steps finish** steps allow you to pause downstream steps until previous steps have completed. This may be necessary for the logic to function properly, but it may increase the overall time for the transformation to complete. It also requires more memory because the row buffers will fill up with results from all rows before proceeding.
- **Parallel Transformations**– PDI job entries normally run sequentially. You can configure the job to run two or more transformations in parallel.
- **Transformation Combining**– Combining two or more transformations into a single transformation will run all steps in parallel.

Memory Utilization

Numerous PDI transformation steps allow you to control how memory is utilized. Allocating more memory to PDI in conjunction with the **fine-tuning** step settings can have a significant impact on performance, for example:

Creating a Row Buffer

- **Recommendation:** Create a row buffer between each step.
- **Rationale:** Row buffers are stored in memory, so this setting allows you to increase or decrease memory used for the buffers.
- **Solution:** Configure the size of the buffer (in rows) by going to **Transformation settings, Miscellaneous** tab, and modify the **Nr of rows in rowset** setting.

Sorting Rows

- **Recommendation:** Sort the rows in memory.
- **Rationale:** Sorting all rows in memory is significantly faster than using a memory-plus-disk approach.
- **Solution:** Use the **Sort size (rows in memory)** setting on the **Sort rows** step to control this. **The Free memory threshold (in %)** helps avoid filling up available memory. Be sure to allocate enough RAM to PDI. The **Compress TMP Files** setting can also conserve memory at the cost of CPU resources.

Joins and Lookups Steps

- **Recommendation:** Use joins and lookup steps to configure data caching.
- **Rationale:** It reduces the number of database queries and improves performance at the cost of using more memory.
- **Solution:** Use the configuration settings to control cache size.

PDI Clusters

PDI clusters allow you to distribute a PDI transformation across multiple servers to leverage more CPU, memory, and network resources. This section provides an overview of the many ways clusters can be used to improve performance.



Figure 6: PDI Cluster

The following is a list of things to keep in mind when working with PDI clusters:

PDI Deployment

- PDI clusters are completely software based and can be deployed in several ways, including physical servers, virtual servers, on premise or in the cloud, or on a Hadoop cluster using the **Start/Stop a YARN Kettle Cluster** steps.

Master and Slave Services

- PDI clusters consist of a master service and one or more slave services. Each master and slave service is implemented by a light-weight web service called Carte.
- Multiple slave services can run on the same server as the master (master server) or on separate servers (node servers).
- A master service can be configured to be dynamic. This allows new slave services to self-register with the master service without interrupting transformations currently running on the cluster.
- Currently running transformations will continue to run on the same set of slave services. Subsequent transformations will take advantage of all the nodes in the cluster.
- The master service will monitor tech servers and will remove slave services that are not available after a threshold has been met.

AWS Auto Scaling

- AWS Auto Scaling can be implemented to allow you to increase or decrease the size of the cluster based on conditions that you define.



Transformations currently running on a disabled node will fail.

Clustered Transformations

There are several ways to work with clustered transformations in PDI. The following figure is an example of a clustered transformation when the **calc elapsed time** and **out_read_lineitems** steps are run on the master and the other steps are run on the nodes:

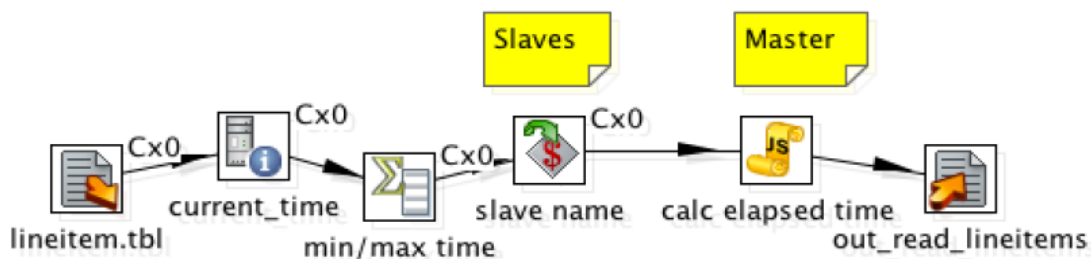


Figure 7: Clustered Transformation

The following table presents techniques for working within clustered transformations:

Table 11: Clustered Transformation Techniques

Technique	Definition
Enable Clustering	You can enable clustering in a transformation job entry by setting the Run this transformation in a clustered mode setting.
Non-clustered Mode	All transformation steps are configured to run in non-clustered mode. Non-clustered steps only run on the master service.
Clustered Mode	Individual transformation steps may be configured to run in clustered mode. These steps will be distributed and run across slave services. A clustered transformation is one that has at least one step configured to run in clustered mode.
Slave and Master services	Slave services must pass data back to the master service if a non-clustered step follows a clustered step. You should try to run as many steps as possible in clustered mode to this. Ideally, all steps are clustered allowing the entire transformation to run in parallel across node servers with no cross-traffic.
Scaling Servers	You should first scale up servers and then develop a PDI cluster due to the additional complexity involved in designing a clustered transformation. Not all transformations are well suited to clustering, but all transformations are multithreaded and may be scaled up easily on a single server.

Partitioning

The following is a list of things to keep in mind when partitioning data within the cluster:

Partitioning Scheme

- A database table may be partitioned across the cluster using a partitioning scheme. For example, a customer table may be partitioned across a three-node cluster based on the APAC, EMEA, and AMER regions.
- PDI allows you to create a corresponding partition scheme that maps to the clustered database connection.
- The partition scheme can be applied to clustered transformation steps. This allows for creating efficient data pipelines. These pipelines are called swim lanes, and they map specific PDI node servers to specific database nodes. This reduces cross-traffic and maximizes network throughput.

Look up Data

- Lookup data can also be divided across node servers based on a partition scheme. For example, each PDI node can pre-load a cache of lookup records that are specific to the data being processed by that node.

Parallel Reads and Writes

- Partitions can be used for parallel reads from data sources or writes to data targets.
- Partitions may also be used for reading multiple text files into multiple PDI nodes, whereby each node can be aware of the files and data it is working with. Some NAS storage clusters implement a headless configuration that allows simultaneous, parallel connectivity to each node in the cluster. This allows for swim lanes that map to PDI clusters for reading and writing files.

Database Clusters

- Database clusters allow large data sets to be stored across multiple servers.
- You can configure a database connection to include the list of database servers that make up the cluster in PDI. This allows PDI nodes to connect to separate database nodes providing maximal network throughput.

The following figure is an example of a fully distributed cluster:

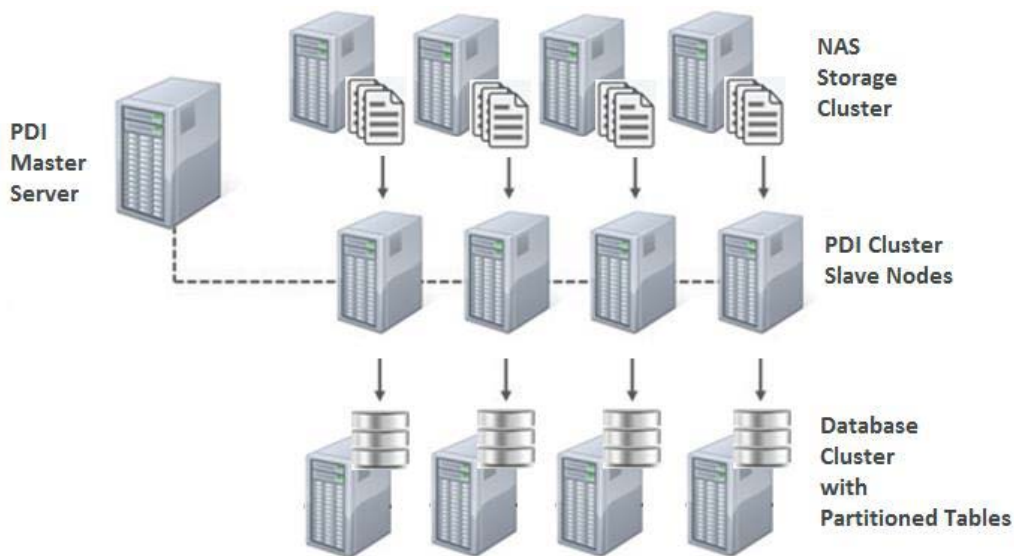


Figure 8: Distributed Cluster

The following table lists the components of the distributed cluster:

Table 12: Distributed Cluster Components

Component	Definition
NAS Storage Cluster	A clustered transformation will extract customer data stored in files on a NAS storage cluster and load it into a database cluster.
Database Cluster with Partitioned Tables	The customer table is partitioned by customer name on the database cluster. The PDI transformation has the database cluster configured and the partition scheme mapped.
PDI Nodes	Each PDI node will connect to a specific database node and will extract those files on the NAS that correspond to the table partition that exists on the database node.

Each PDI node connects to a separate node on the NAS storage cluster and a separate node on the database cluster. This provides maximum network efficiency by creating swim lanes from the source, through PDI, to the data target. The partition scheme ensures there is no cross-traffic.

PDI Visual MapReduce for Hadoop

PDI allows you to deploy transformations as MapReduce jobs on a Hadoop cluster in a process called Visual MapReduce. This section illustrates some of the advantages of using this approach.

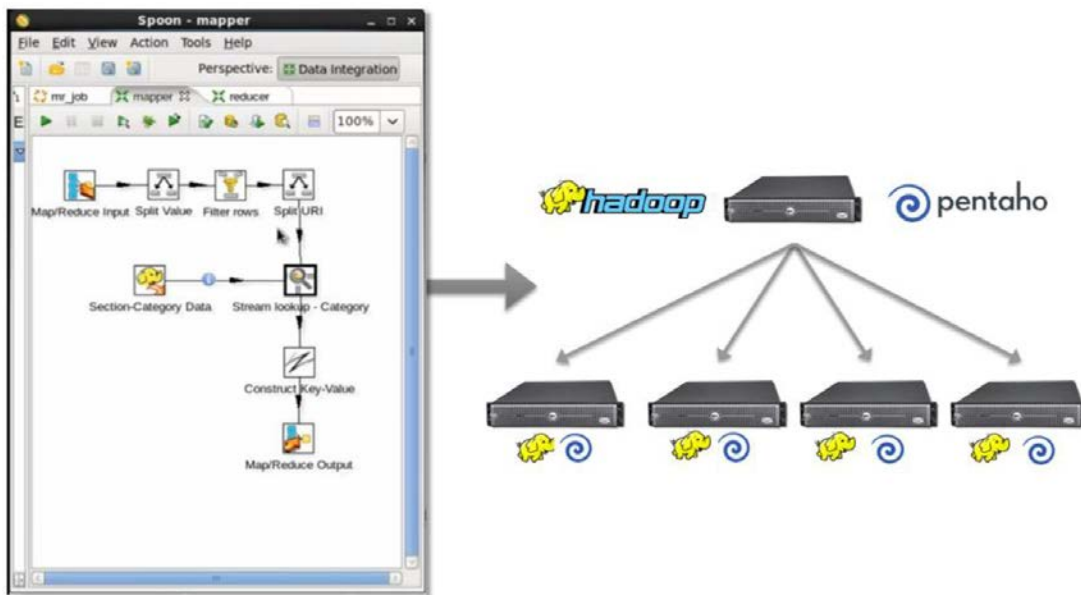


Figure 9: Hadoop Visual MapReduce

The Visual MapReduce process allows you to create MapReduce jobs using the same visual, drag-n-drop interface and zero-code transformation steps used for creating regular PDI transformations.

The following is a list things to keep in mind when working with Visual MapReduce:

PDI Connectivity

- PDI uses an adaptive big data layer called shim. PDI works with several Hadoop distributions, and shim works as a translator, providing a transparent, consistent interface for working with Hadoop.

Mapper, Combiner, and Reducer

- Visual MapReduce involves a mapper transformation (required), combiner transformation (optional), reducer transformation (optional), and a PDI job with a **Pentaho MapReduce** step. The step must be configured to connect to a Hadoop cluster, and then submitted to a MapReduce job using the mapper, combiner, and reducer transformations.
- Mapper, combiner, and reducer transformations are PDI transformations with a **MapReduce Input** step, a **MapReduce Output** step, and any number of other transformation steps that transform data.

Pentaho MapReduce Job Entry

- The PDI engine will automatically deploy to each node in the Hadoop cluster when a **Pentaho MapReduce** job entry is executed.
- The **Pentaho MapReduce** job will run on each node in the cluster working with local data.



There is no special configuration required on the Hadoop cluster because PDI will take care of deploying the PDI engine. Subsequent Pentaho MapReduce jobs will not require deploying the engine as it remains cached on the Hadoop cluster.

Related Information

Please visit the following links for more information about topics discussed in this document:

Best Practice Documentation

- [Pentaho and Amazon Web Services](#)
- [Performance Monitoring and Logging](#)

Pentaho Documentation

- [PDI Performance Tuning Tips](#)
- [Pentaho Hardware and Software Requirements](#)

Pentaho Wiki

- [Partitioning data with PDI](#)
- [Pentaho Community Wiki](#)
- [Table Output Step](#)

Amazon

- [Amazon S3](#)
- [Amazon EC2 Instance Types](#)
- [Amazon EC2 Instance Store](#)

Additional Resources

- [ExpeDat](#)
- [WAN optimization](#)
- [Sneakernet](#)