Best Practice - Pentaho Data Integration Performance Tuning

by Rodrigo Haces - Tuesday at 00:54

Introduction

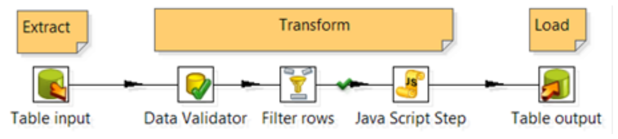
This guide provides a high-level overview of factors that can affect performance of Pentaho Data Integration (PDI) jobs and transformations and provides a methodical approach to identifying and addressing bottlenecks.

Performance Tuning Process

Overview

PDI transformations are ETL workflows that consist of a number of steps linked together as show below. 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 Step** steps).
- Output step – outputs transformed data from PDI (e.g. **Table output** step).



All steps are processed in parallel. The overall speed or throughput 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 SLA is met.

Internal vs. External Performance Factors

If the bottleneck is an input or output step then it is likely due to something external to PDI, such as network or database performance. If the bottleneck is a transformation step then it is likely due to PDI performance. Many factors can affect performance and this guide groups them into two sections: **External Performance Factors** and **PDI Performance Tuning**.

Identifying Bottlenecks

Several PDI features assist with identifying bottlenecks.

Step Metrics

A row buffer is created between each step. Steps retrieve rows of data from their inbound row buffer, process the rows, and pass them into an outbound row buffer which feeds into the subsequent step. By default, row buffers can hold up to 10,000 rows, but this can be configured for each transformation.

When you run a transformation, the **Step Metrics** tab on the **Execution Results** pane shows real-time statistics for each step. 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. If the input buffer of a step is full then you know that the step cannot keep up with the rows being fed into it. Below are some examples.

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 example above shows a snapshot in time of a running transformation. Since all of the buffers are low (much less than 10,000) it is clear that the transformation has no trouble keeping up with incoming rows, processing them and delivering to the target. Therefore, the Table input step is the bottleneck.

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

The example above shows that all the buffers are full (close to the buffer size of 10,000). This means that PDI is waiting for the Table output step to consume rows. The data target is the bottleneck.

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 example above shows that the row buffers are filled all the way through to the Java Script Step. The Table output buffers are low which indicates that the data target has no trouble consuming output from PDI. This indicates that the Java Script Step is the bottleneck.

Eliminating the bottleneck

If an input step or output step is the bottleneck, follow the guidelines in the **External Performance Factors** section below to address areas such as networking, database, and storage optimization that can affect how quickly data can be imported/exported from PDI. Otherwise, go to the **PDI Performance Tuning** section to improve performance within PDI.

The **Metrics** tab on the **Execution Results** pane shows the length of time in milliseconds for initialization and execution of each transformation step and can also assist with identifying bottlenecks.

Performance Monitoring

Real-time performance monitoring captures throughput in rows per second over time for each step for several metrics. Performance monitoring values can be stored in a centralized logging database to enable analyzing jobs run on remote PDI servers. Performance monitoring does require some additional resources and can be enabled/disabled on individual jobs and transformations.

See http://help.pentaho.com/Documentation/5.4/0L0/0Y0/070 for additional information on Performance Monitoring and Logging.

Repeat Measurements

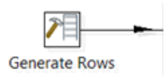
Data caching can significantly affect performance for subsequent runs. For example, a database may cache query results so that the next time the query is run it will return results much faster. Therefore, it is best practice to measure the same scenario several times to account for caching.

Verifying Bottlenecks

You can verify that the bottleneck is an output step by replacing the output step with a dummy step which just throws away the rows. If the overall speed of the transformation increases then it is likely that the output step is the bottleneck.

Dummy (do nothing)

For input step bottlenecks, you can replace the input step with a **Generate Rows** step, **Data Grid** step, or a **Text file input** step pointing to a file on fast local storage or a RAM drive. Check if the transformation is faster.



If the bottleneck step is a transformation step, follow guidelines in the **PDI Performance Tuning** section below to improve the performance of PDI.

External Performance Factors

PDI is, by its nature, part of a larger system that includes data sources, data targets, networking, storage, and other infrastructure components. This guide discusses these areas at a high level 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 no matter how much hardware you throw at your database or PDI server, throughput is capped by the network layer. The following techniques can assist with addressing network bottlenecks:

- **Diagnosis** - determine if the network is the bottleneck by eliminating it altogether. Export source data to a local text file and measure how long it takes for the database to export the data without touching the network. Then copy the text file to the PDI server and measure the time for the transfer. Modify the transformation to import the local file. Then run the transformation again and measure the time to import the local file without touching the network. Compare these measurements to assess network performance.

- **Network sizing** – Consider adding additional NIC's or upgrading to 10gbps. Scale out data sources/targets and/or PDI using clustering technology which leverages network connectivity across multiple servers (see below for more on PDI clusters). 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 WAN presents a number of challenges. Consider moving data sources, data targets, and/or PDI servers to be on the same LAN. If you must move data across a WAN there are a number of techniques and third-party appliances designed to improve throughput. One alternative to direct database connections is to dump data to a text file and performing a file transfer using a WAN optimized tool such as ExpeDat (http://www.dataexpedition.com/expedat) See http://en.wikipedia.org/wiki/WAN_optimization for more info.
- **Cloud Computing** – network configuration in the cloud can be tricky due to the lack of transparency of the implementation. Some things to consider with AWS include (see http://aws.amazon.com/ec2/instance-types/):
- Ephemeral storage – use local attached SSD storage instead of EBS which sits on a NAS. See below for **Storage Performance** section below for data loss considerations.
- Provisioned IOPS
- Enhanced Networking
- Cluster Networking
- **Offline shipping** – in extreme cases it can be faster to overnight hard drives to far off locations, avoiding the network altogether. Large data migration efforts are a common use case for offline shipping. See http://en.wikipedia.org/wiki/Sneakernet .

Data Source/Target Performance

If the network is not the bottleneck then it may be the performance of the data source or target. Database optimization is an advanced topic with many techniques that vary greatly from one product to the next. Some common approaches are listed below.

- **Query optimization** – this is almost always the first place to start. Many databases provide a SET EXPLAIN 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 /dev/null and measure throughput. This may represent the fastest throughput possible for the data source/target.
- **Bulk Loaders** – many databases provide bulk loaders that may be faster than performing insert queries. PDI includes bulk loader transformation steps for a number of databases. PDI also supports calling command-line bulk loaders.
- **Data Staging/Pre-processing** – consider creating a materialized view or pre-processing data on the database and/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 may provide better performance for certain situations.
- **Replication** – database replication may allow a mirror image of a database to be kept physically close to PDI. This can reduce or even eliminate network connectivity between PDI and the data source/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 that can improve performance for certain types of queries. PDI has functionality for configuring transformations to leverage these features. See the **PDI Clusters** section below for more information on how to leverage these features.

Storage Performance

Data storage outside of a database can be necessary for working with data files, staging, batching, archive, etc. Use the following table as guide for choosing the correct storage option. Note: throughput (MB/s) shown are rough estimates.

Solution	Approx. MB/S	Description
RAM disk	17,000	RAM is the fastest hardware storage option. The OS can be configured to cache files in RAM. RAM drives are easily created in Linux/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/processing. RAM is expensive, volatile (erased on reboot), and limited in capacity
SSD	2,000	Solid State Drives provide fast, non-volatile (permanent) storage mounted as a local hard drive. These can come in the form of a PCIe card installed on the server motherboard. SSD's also provide fast random access compared to rotational media
HDD (physical)	750	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.
	Depends on	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 a NAS or SAN but can be local attached storage. Performance, cost, capacity and other specs depend on multiple factors, including: cost of storage server, capacity limits imposed

vHDD (virtual HDD)	type	by file system or VMware or other cloud infrastructure, speed of networking and storage server, load on shared resources, etc. vHDDs can be thin-provisioned, i.e. a 100GB vHDD with 10GB data on it will only occupy 10GB on backend storage but will appear as 100GB to the VM's OS. vHDDs can also be expanded more easily than physical storage. In some cases LVM can support expansion of a vHDD with zero down time.
NAS/SAN	30	A NAS/SAN provides fail-over, redundancy and (optionally) disaster recovery, offsite backup, huge capacity, etc. These typically provide access via NFS, CIFS, or iSCSI. Third party vendors can provide local NAS/SAN storage inside the data centers of cloud providers such as AWS. This can provide a high performance alternative to S3 and EBS.
AWS Ephemeral Storage	Depends on type	Local storage (HDD or SSD) attached to the host. Considered temporary as data will be lost if the server is shutdown. See http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html
AWS EBS	30-125	EBS is vHDD provided by AWS (see vHDD above). Approx. 10x more durable than physical HDD due to replication on back-end NAS. Snapshots and/or RAID10 are still recommended. Striping EBS volumes can increase performance and capacity.
AWS S3	1	Eventual consistency model is a major factor affecting usage (see Amazon Simple Storage Service (S3) for details). S3 performance is also much slower than EBS.
AWS Glacier	See description	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 complete within 3-5 hours. Upload/download speeds may be similar to S3.

PDI Performance Tuning

This section provides techniques for tuning various aspects of PDI, including:

- PDI Transformation Design
- PDI Job Design
- Scaling Up Transformations
- PDI Clusters
- PDI Visual Map/Reduce for Hadoop

In general, it makes sense to start with optimizing ETL to be as efficient as possible and then evaluate platform-related factors such as hardware sizing, clustering, etc.

PDI Transformation Design

Data Caching

High network latency can make executing multiple queries much slower than running a single bulk query. Most lookup steps allow you cache lookup values. You can even perform up-front loading of records in a single query and cache all 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. See documentation on the table output step for additional info around using batch updates.

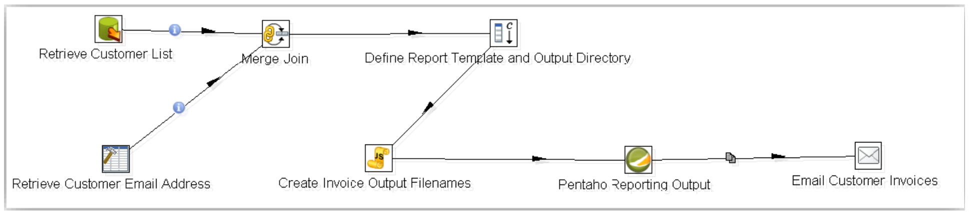
Database Processing

- **Database Sorting:** Sorting on the database is often faster than sorting externally, especially if there is an index on the sort field(s). See the **Memory** section below for configuring the **Sort rows** step to leverage memory and CPU resources.
- **Database Processing:** In some cases, performance will be better if data is processed directly on the database. This approach may completely 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 Extract/Load/Transform (ELT) design pattern. Stored procedures, triggers, materialized views, and aggregation tables are just some of the techniques that can be leveraged. See **Data Source/Target Performance** section above for more info.
- **Prepared Statements:** Most database steps prepare statements which incur some overhead up front but improve 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.

Real-time data on demand

- **Extract/Transform/Report (ETR):** PDI transformations also support Extract/Transform/Report (ETR). Pentaho reports and the dashboard framework can use PDI transformations as a native data source.
- **Report Bursting:** A PDI transformation can feed results into a PDI report template and burst

the report out via email or to a file server without having to stage the data in a reporting table. Below is an example of report bursting with a PDI transformation.


Retrieve Customer List
Merge Join
Define Report Template and Output Directory
Retrieve Customer Email Address
Create Invoice Output Filenames
Pentaho Reporting Output
Email Customer Invoices

- **PDI thin JDBC driver:** The PDI JDBC driver provides a data on-demand interface. Any application that can connect to an ODBC or JDBC data source can send an SQL query to a PDI transformation via 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 however it may not perform as well as other highly optimized, single-purpose transformation steps.
- Turn off compatibility mode when not needed. This will run the newer, faster JavaScript engine.
- Consider writing a step plugin. This can provide better performance than using a JavaScript step.
- A **User Defined Java Class** step may perform better than a JavaScript step.

Constant/Static Values

Avoid calculating the same static value on every row. Instead you can calculate constants in a separate transformation and set variables to be used in downstream transformations. Or you can calculate constants in a separate stream and use the **Join Rows (Cartesian product)** step to join the constant into the main stream.

Text files - CSV file input, Text file input, Text file output transformation steps

- The **Lazy conversion** setting will postpone data conversion as long as possible. This includes character decoding, data conversion, trimming, etc. This can be helpful if not all fields are used, if data will be simply written out to another text file, or in some bulk loader scenarios.
- The **NIO buffer size** parameter determines the amount of data read at once from a text file. This can be adjusted to increase throughput.

Logging and Performance Monitoring

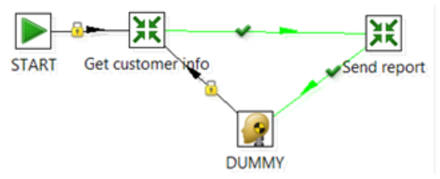
Detailed logging and performance monitoring can be very helpful in development and test environments. The logging level can be turned down and performance monitoring can be disabled for production environments to conserve resources.

See http://help.pentaho.com/Documentation/5.4/0L0/0Y0/070/030 for additional Pentaho Data Integration Performance Tuning Tips.

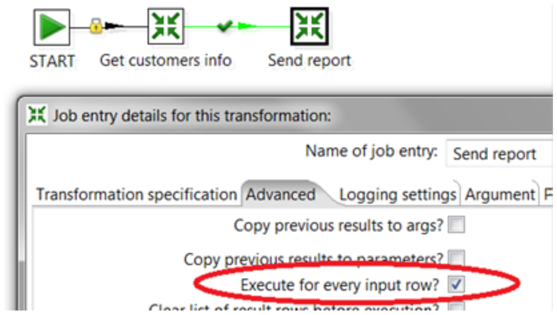
PDI Job Design

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 which then loops back to **Get customer** which retrieves the next customer and the loop continues until there is no data left.



Instead of creating a loop in a job, set the **Execute for every input row** setting.



Job entry details for this transformation:

Name of job entry: Send report

Transformation specification | Advanced | Logging settings | Argument

Copy previous results to args? ☐

Copy previous results to parameters? ☐

Execute for every input row? ☑

The **Get customers info** will retrieve all 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.

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.

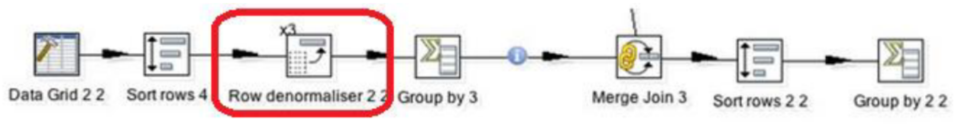
Scaling Up Transformations

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

CPU and Multithreading

PDI transformations are multithreaded. Each step in a transformation gets its own thread. Transformation steps run in parallel, leveraging multiple cores.

- You can increase the number of copies of a step to increase threads assigned to the step. This allows you to assign more CPU resources to slower steps. In the example below, the **Row denormalizer** step is assigned three step copies which will spawn three threads. Each of the other steps will spawn its own thread. Therefore, transformation will spawn a total of 9 threads which could utilize up to 9 cores on the PDI server.



  Be careful not to allocate too many threads as this can actually degrade performance. As a general rule, optimal performance is achieved by keeping the number of steps to less than 3-4 times the number of cores.

- 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 as well as require more memory as the row buffers will fill up with results from all rows before proceeding.
- PDI job entries normally run sequentially. You can configure the job to run two or more transformations in parallel.
- Combining two or more transformations into a single transformation will run all steps in parallel.

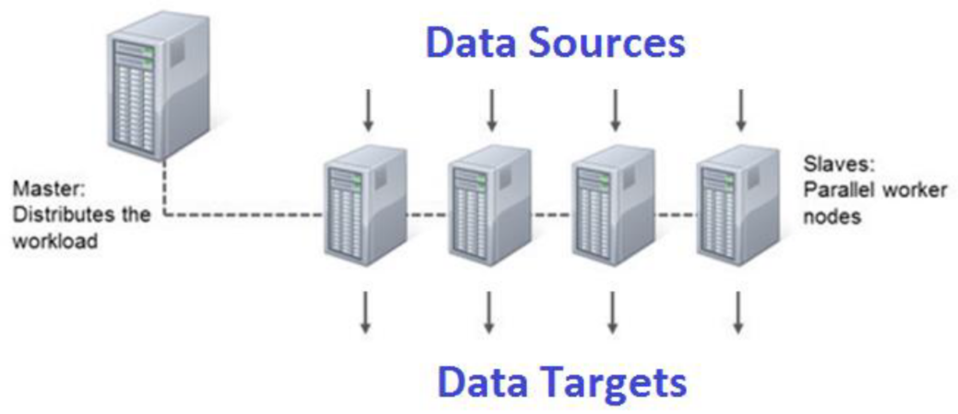
Memory Utilization

Many PDI transformation steps allow you to control how memory is utilized. Allocating more memory to PDI in conjunction with fine tuning step settings can have a big impact on performance.

- A row buffer is created between each step. You can configure the size of the buffer (in rows): go to **Transformation settings**, **Miscellaneous** tab, and modify the **Nr of rows in rowset** Row buffers are stored in memory so this setting allows you to increase or decrease memory used for the buffers.
- Sorting all rows in memory is significantly faster than using a memory-plus-disk approach. 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.
- Several join and lookup steps allow you to configure data caching which can reduce the number of database queries. This can improve performance at the cost of using more memory. Some configuration settings allow you 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 guide does not provide detailed instructions on setting up a PDI cluster but instead is a general overview of functionality and a description of different ways clusters can be utilized to improve performance.


Data Sources
Master: Distributes the workload
Slaves: Parallel worker nodes
Data Targets

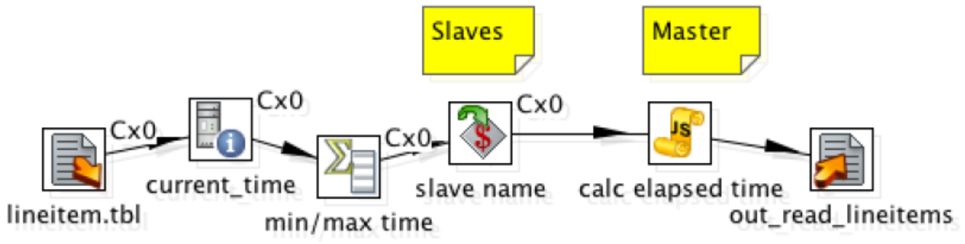
Overview

- PDI clusters are completely software based and can be deployed in several ways:
- Physical servers
- Virtual servers, on premise or in the cloud.
- As of version 5.1 you can spin up a PDI cluster on a Hadoop cluster using the new **Start/Stop a YARN Kettle Cluster**
- 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 (slave 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 all slaves in the cluster.
- AWS Auto Scaling can be implemented to allow you to increase or decrease the size of the cluster based on conditions that you define.
- The master service will monitor slave servers and will remove slave services that are not available after a threshold has been met. Note that transformations currently running on a slave that is disabled will fail.

Clustered Transformations

- You enable clustering in a transformation job entry by setting the **Run this transformation in a clustered mode**
- By default, all transformation steps are configured to run in non-clustered mode. Non-clustered steps only run on the master service.
- 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.
- If a non-clustered step follows a clustered step, the slave services must pass data back to the master service. In general, try to run as many steps as possible in clustered mode to avoid passing data back and forth between the master and the slave services. Ideally, all steps are clustered allowing the entire transformation to run in parallel across slave servers with no cross-traffic.

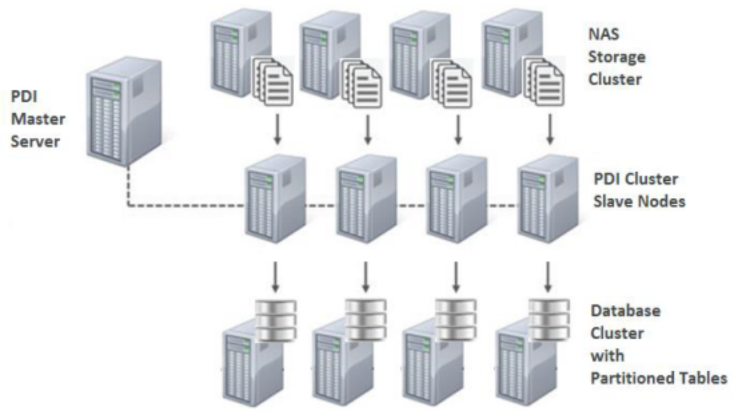
- In the example below, the **calc elapsed time** and **out_read_lineitems** steps will run on the master, all others steps will run on the slaves.



- Due to the additional complexity involved in designing a clustered transformation, it is often preferable to first scale up servers and then pursue developing a PDI Cluster. Furthermore, not all transformations are well suited to clustering but all transformations are multithreaded and may be scaled up easily on a single server.

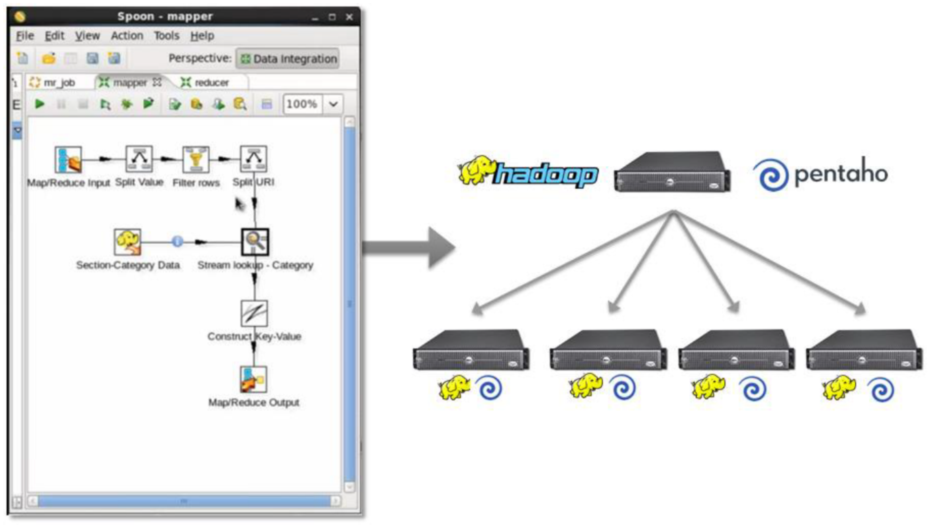
Partitioning

- Database clusters allow large data sets to be stored across multiple servers.
- In PDI you can configure a database connection to include the list of database servers that make up the cluster. This allows PDI slaves to connect to separate database nodes providing maximal network throughput.
- A particular 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 region: APAC, EMEA, and AMER.
- 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, so-called “swim lanes”, that map specific PDI slave servers to specific database nodes. This reduces cross-traffic and maximizes network throughput.
- 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 slaves whereby each slave 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 allowing for swim lanes that map to PDI clusters for reading and writing files.
- Lookup data can also be divided across slave servers based on a partition scheme. For example, each PDI slave node can pre-load a cache of lookup records that are specific to the data being processed by that node.
- The example below shows a fully distributed system.
- A clustered transformation will extract customer data stored in files on a NAS storage cluster and load it into a database cluster.
- 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.
- Each PDI Slave 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 slave 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 guide does not provide detailed instructions for implementing Visual MapReduce but serves to illustrate some of the advantages of using this approach.



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.

- PDI has broad connectivity for a number of Hadoop distributions. PDI includes "shims" that abstract away the particulars of specific distributions providing a consistent interface for working with Hadoop. This is part of the Pentaho Adaptive Big Data Layer.
- Visual MapReduce involves the following components:
- Mapper transformation (required)
- Combiner transformation (optional)
- Reducer transformation (optional)
- PDI job with a **Pentaho MapReduce** step that is configured to connect to a Hadoop cluster and submit 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 in between that transform the data.
- When a **Pentaho MapReduce** job entry is executed the PDI engine will be automatically deployed to each node in the Hadoop cluster. Note that there is no special configuration required on the Hadoop cluster as 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.

- The **Pentaho MapReduce** job will run on each node in the cluster working with local data.

References

Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration By Matt Casters, Roland Bouman, and Jos van Dongen

Pentaho Documentation: http://help.pentaho.com

Pentaho Infocenter (legacy documentation): http://infocenter.pentaho.com

Pentaho Community: http://community.pentaho.com

Pentaho Community Wiki: http://wiki.pentaho.com

Pentaho hardware and software requirements:
http://help.pentaho.com/Documentation/5.4/0D0/160/000

Performance Monitoring and Logging: http://help.pentaho.com/Documentation/5.4/0L0/0Y0/070

Partitioning: http://wiki.pentaho.com/display/EAI/Partitioning+data+with+PDI

Dynamic Clusters: http://wiki.pentaho.com/display/EAI/Dynamic+clusters

WAN Optimization: http://en.wikipedia.org/wiki/WAN_optimization

PDI Scale Out Whitepaper:
http://www.bayontechnologies.com/bt/ourwork/pdi_scale_out_whitepaper.php

Privacy Policy | Legal Notices | Safe Harbor Privacy Policy