



# Performance Tuning for Pentaho Data Integration (PDI)

# HITACHI

## Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

# Contents

- Overview..... 1
  - Before You Begin..... 1
    - PDI Workflow ..... 1
- Identify, Eliminate, and Verify Bottlenecks ..... 2
  - Identifying Bottlenecks with Step Metrics ..... 2
    - Example of Input Bottleneck ..... 2
    - Example of Output Bottleneck..... 3
    - Example of Transformation Bottleneck..... 3
  - Eliminating Bottlenecks..... 3
  - Verifying Bottlenecks ..... 4
- External Performance Factors ..... 5
  - Network Performance ..... 5
  - Data Source and Target Performance..... 6
  - Storage Performance..... 7
- PDI Performance Tuning ..... 9
  - PDI Transformation Design ..... 9
    - Query Management..... 9
    - Database Management..... 10
    - Realtime Data on Demand ..... 10
    - Scripting..... 11
    - Manage Thread Priorities ..... 11
    - Other Performance Optimization Options ..... 12
  - PDI Job Design ..... 13
    - Avoiding Loops ..... 13
- Scaling Up Transformations ..... 14
  - CPU and Multithreading..... 14
  - Memory Utilization ..... 15
- Scaling Out Transformations..... 16
- Other Related Information..... 17
- Finalization Checklist..... 17



## Overview

This document covers some best practices on factors that can affect the performance of Pentaho Data Integration (PDI) jobs and transformations. You will learn a methodical approach to identifying and addressing bottlenecks in PDI.

Our intended audience is PDI administrators who are interested in maximizing PDI performance.

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.

## Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document. This document assumes that you have some knowledge of PDI and have already installed Pentaho.

### *PDI Workflow*

PDI transformations are extract, transform, and load (ETL) workflows that consist of steps linked together as shown:

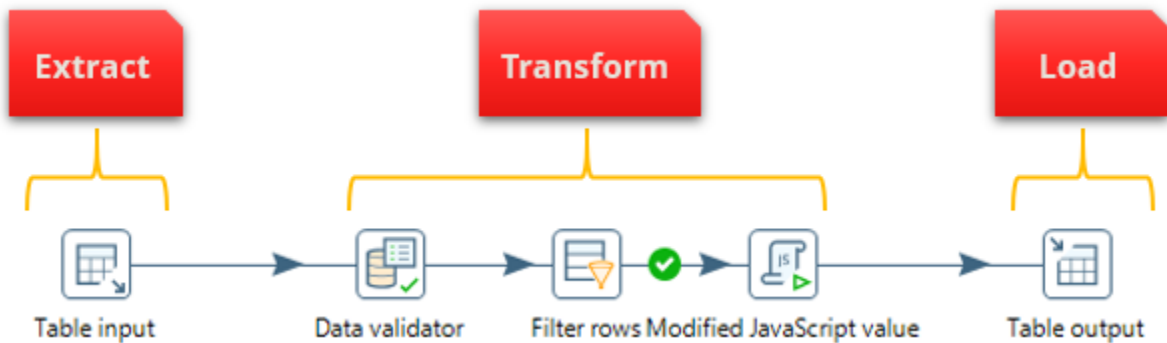


Figure 1: Sample PDI ETL Workflow

There are three basic types of steps:

1. **Input step:** Ingests data into PDI (for example, the **Table input** step)
2. **Transformation step:** Processes data within PDI (for example, the **Data Validator**, **Filter rows**, and **Modified JavaScript value** steps)
3. **Output step:** Outputs transformed data from PDI (for example, the **Table output** step)

# Identify, Eliminate, and Verify Bottlenecks

Within PDI, 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 and iterate until the service level agreement (SLA) is met.

More information is available in the following sections of this document:

- [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. Then, the rows are 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 realtime statistics for each step when you run a transformation. The Input/Output field shows a realtime 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 Input Bottleneck*

This example shows a snapshot in real time of a running transformation:

*Table 1: Input Bottleneck*

Step Name	Input/Output
<b>Table input</b>	0 / 50
<b>Data validator</b>	48 / 52
<b>Filter rows</b>	54 / 42
<b>Modified JavaScript value</b>	37 / 51
<b>Table output</b>	43 / 0

In this instance, the **Table input** step is the bottleneck because the buffers are so low (much less than 10,000) for the downstream steps.

### Example of Output Bottleneck

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

Table 2: Output Bottleneck

Step Name	Input/Output
Table input	0/9720
Data validator	9850/9741
Filter rows	9922/9413
Modified JavaScript value	9212/9413
Table output	9985/0

In this case, PDI is waiting for the **Table output** step to consume rows. This means that the data target (**Table output**) is the bottleneck.

### Example of Transformation Bottleneck

This example shows that the row buffers are filled all the way through to the **Modified JavaScript value** step:

Table 3: Transformation Bottleneck

Step Name	Input/Output
Table input	0/9815
Data validator	9784/9962
Filter rows	9834/9724
Modified JavaScript value	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 Modified JavaScript value step is the bottleneck.

## Eliminating Bottlenecks

Consider the following actions to detect and eliminate bottlenecks:

Table 4: Eliminating Bottlenecks

Solution	Explanation
Performance Monitoring	Realtime <a href="#">performance monitoring</a> 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 services. Performance monitoring requires additional resources and can be enabled or disabled on individual jobs and transformations.

Solution	Explanation
<b>Repeat Measurements</b>	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 of this document 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 to or exported from PDI. Otherwise, the [PDI Performance Tuning](#) section gives suggestions for improving performance within PDI.



We recommend selecting the **Metrics** tab (different from the **Step Metrics** tab) on the **Execution Results** pane to view the length of time in milliseconds for the initialization and execution of each transformation step. This can help you identify bottlenecks.

## Verifying Bottlenecks

You can verify that the bottleneck is an output step by replacing it with a **Dummy (do nothing)** step, which throws away the rows. It is likely that the output step is the bottleneck if the overall speed of the transformation increases when you use this method.

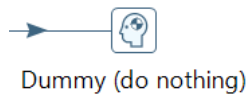


Figure 2: Verifying Bottleneck as Output

You can replace an input step with a Generate rows step, a 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.

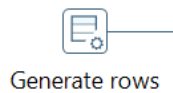


Figure 3: Verifying Bottleneck as Input

Follow the guidelines in the [PDI Performance Tuning](#) section if the bottleneck step is a transformation step.



## 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.

More information can be found on these topics in the following sections:

- [Network Performance](#)
- [Data Source and Target Performance](#)
- [Storage Performance](#)

## 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 methods to assess for possible network bottlenecks:

*Table 5: Network Bottleneck Troubleshooting*

Solution	Explanation
<b>Network Sizing</b>	Consider adding additional network interface controllers (NICs) or upgrading to 10Gbps. Scale out data sources, targets, and PDI using clustering technology, which optimizes network connectivity across multiple servers. Ethernet bonding can provide increased throughput as well as failover.
<b>Network Hardware</b>	Switches, routers, proxy servers, firewalls, and other network appliances can create bottlenecks. Consider upgrading or bypassing these altogether.
<b>WAN Optimization</b>	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.

Solution	Explanation
<b>Cloud Computing</b>	Network configuration in the cloud can cause issues due to the lack of transparency in the implementation. Networking can also play a strong role in cloud computing performance if you have your database in a noncloud environment and your Pentaho Server in a cloud environment, or vice versa. More information on this is available at <a href="#">Amazon EC2 Instance Types</a> .
<b>Colocation</b>	Moving your database and Pentaho Server physically close together in the same facility will keep network latency from being as much of a factor.
<b>Robust Storage</b>	Use a local attached solid state drive (SSD) instead of an elastic block store (EBS), which sits on a network-attached storage (NAS). The <a href="#">Storage Performance</a> section of this document has more on data loss considerations.
<b>Offline Shipping</b>	In extreme cases, it is faster to physically ship hard drives overnight to far-off locations, avoiding the network altogether. Large data migration efforts commonly make use of 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. Here are some of the more common approaches to database optimization:

Table 6: Database Optimization Techniques

Solution	Explanation
<b>Query Optimization</b>	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.
<b>Local Export/Import</b>	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.
<b>Bulk Loaders</b>	Many databases provide bulk loaders that may be faster than performing <code>insert</code> queries. PDI includes bulk loader transformation steps for several databases. PDI also supports calling command line bulk loaders.
<b>Data Staging/Preprocessing</b>	Consider creating a materialized view, preprocessing data on the database, or loading a staging table. These approaches can simplify the ETL logic and possibly reduce data volume over the network.
<b>Database Technologies</b>	Hadoop, NoSQL, analytical, multidimensional, in-memory, cache, and other database technologies can provide better performance for certain situations.
<b>Replication</b>	Database replication allows a mirror image of a database to be kept close to PDI. Keeping it on the same server or in the same datacenter can reduce or even eliminate the need for network connectivity between PDI and the data source or target.

Solution	Explanation
<b>Database Design</b>	Star schemas and other data mart design patterns can dramatically improve performance at the cost of additional complexity and resources.
<b>Clustering/ Sharding/ Partitioning</b>	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 make use of these features. The <a href="#">PDI Clusters</a> section of this document has more information about using 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	Approximate MB/s	Explanation
<b>RAM Disk</b>	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.
<b>SSD</b>	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.
<b>NAS/SAN</b>	30	NAS and storage area networks (SAN) provide failover, redundancy, and (optionally) disaster recovery, offsite backup, huge capacity, and more. These typically provide access through the network filesystem (NFS), common internet filesystem (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.
<b>AWS EBS</b>	30-125	EBS is provided by AWS. It is approximately ten times more durable than physical hard drives due to replication on backend NAS. Snapshots or RAID 10 is still recommended. Striping EBS volumes can increase performance and capacity.

Solution	Approximate MB/s	Explanation
<b>AWS Glacier</b>	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.
<b>AWS S3</b>	1	S3 provides scalability and easy management, but its performance is much slower than EBS. More information is available at <a href="#">Amazon S3</a> .
<b>vHDD (virtual hard drive)</b>	Depends on type	<p>Virtual hard drives are used by virtual machines (VMs). 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 filesystem, 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 (for example, a 100GB vHDD) with 10GB data. In this example, 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 more easily than physical storage. In some cases, the logical volume manager (LFM) can support expansion of a vHDD with zero downtime.</p>
<b>HDD (physical)</b>	750	The throughput shown is for a single hard drive. RAID configurations can provide redundancy, failover, 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](#)
- [Scaling Out Transformations](#)

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 Techniques*

Technique	Explanation
<b>Data Caching</b>	High network latency and/or slow database performance 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 upfront loading of records in a single query and cache the results, instead of performing multiple queries.
<b>Batch Updates</b>	Batch updates can also reduce the number of queries. The commit size setting controls the size of the batches. More information is available at <a href="#">Table Output</a> .

## Database Management

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

Table 9: Database Management Techniques

Technique	Explanation
<b>Database Sorting</b>	<p>Sorting on the database is often faster than sorting externally, especially if there is an index on the <code>sort</code> field(s). The <a href="#">Memory Utilization</a> section of this document has more on configuring the <b>Sort rows</b> step to make use of memory and CPU resources.</p> <p> Remember that databases can sort differently, and Pentaho can sort differently from how the database might sort. For example, some databases do a case insensitive sort while Pentaho would do a case sensitive sort.</p>
<b>Prepared Statements</b>	<p>Most database steps prepare statements, which incurs some overhead up front but improves performance overall. The <b>Execute SQL script</b>, <b>Execute row SQL script</b>, and <b>Dynamic SQL row</b> steps do not perform this initial statement preparation and may not perform as well.</p>
<b>Database Processing</b>	<p>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 preprocessed or staged on the database to simplify PDI transformations. Transformation logic can also be moved to the target sources in an ETL design pattern. Stored procedures, triggers, materialized views, and aggregation tables are just some of the techniques you can use.</p>

## Realtime 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 fileserver, without having to stage the data in a reporting table.

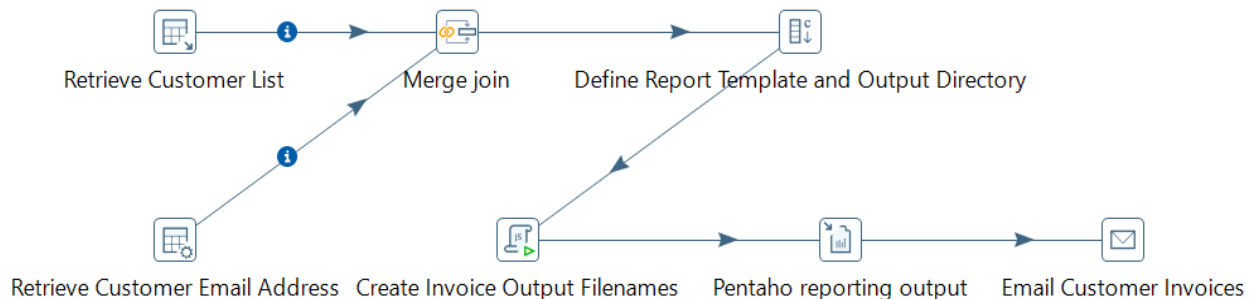


Figure 4: Report Bursting

Some other techniques for viewing data in real time include:

- **Extract, transform, and report (ETR):** PDI transformations support ETR. Pentaho reports and dashboard frameworks can use PDI transformations as a native data source.

- PDI thin JDBC driver:** Any application that connects to a Java database connectivity (JDBC) data source can send an SQL query to a PDI transformation, 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 **Modified JavaScript value** 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 **Modified JavaScript value** step performance:

Table 10: JavaScript Performance Techniques

Technique	Explanation
<b>Compatibility Mode</b>	Turn off compatibility mode when not needed. This will run the newer, faster JavaScript engine.
<b>User defined Java class step</b>	A <b>User defined Java class</b> step may perform better than a <b>Modified JavaScript value</b> step.
<b>Step plugin</b>	Consider writing a step plugin. This can provide better performance than using a JavaScript step.

## Manage Thread Priorities

**Manage thread priorities** is a transformation setting that allows Pentaho to improve performance by reducing locking and lowering CPU usage. This setting is enabled by default in new transformations.

Manage thread priorities is configured in the **Miscellaneous** tab of the transformation's **settings**:

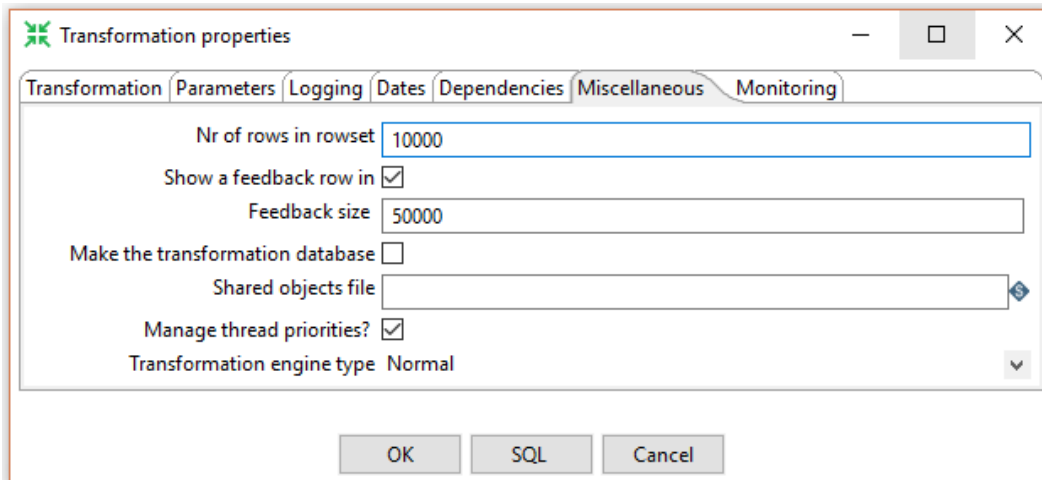


Figure 5: Manage thread priorities

Although the manage thread priorities setting is enabled by default and is designed to improve performance, in one specific case this setting can have a significant negative impact on performance:

when you are merging data from two hops where those two hops are producing records at significantly different throughputs.

Manage thread priorities will only impact performance when you are merging two hops using a step that is neither the **Append streams** step, nor in the **Joins** category in PDI.

Consider the following common example where Manage thread priorities can cause a negative performance impact.

---

*You are processing a 100 million record customer file. When processing this file, you want to evaluate the customer's age and set a flag if they are under 18. Minors only make up 1% of your customer population.*

---

With this requirement, you may use the following pattern in your transformation.

1. **Text file input** step
2. To a **Filter rows** step on customer's age
3. To two **Set field value** steps to set the minor flag
4. Merge these two streams together with a **Dummy** step
5. Write the result to an output file

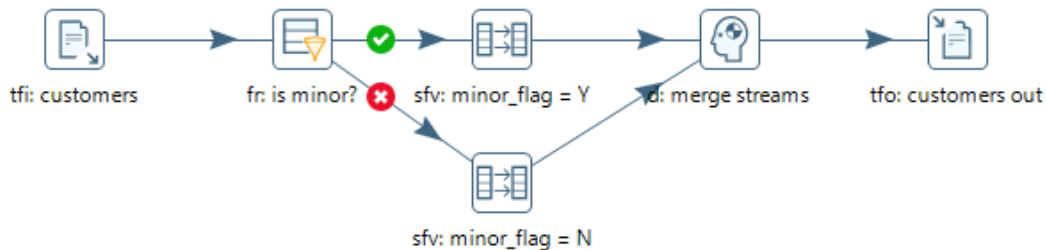


Figure 6: Manage thread priorities Example

Assuming the customer file is randomly distributed on age, the **sfv: minor\_flag = Y** step will produce records at 1% of the throughput of the **sfv: minor\_flag = Y** step. In this situation, it is possible that disabling Manage thread priorities will improve performance of the transformation.



*There are many factors that impact performance. Baseline transformation performance with this setting enabled and compare it to a performance test after disabling this setting.*

## Other Performance Optimization Options

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, including 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 [logging and 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. [Performance Tuning](#) has more information on this topic.

## PDI Job Design

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

### Avoiding Loops

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:

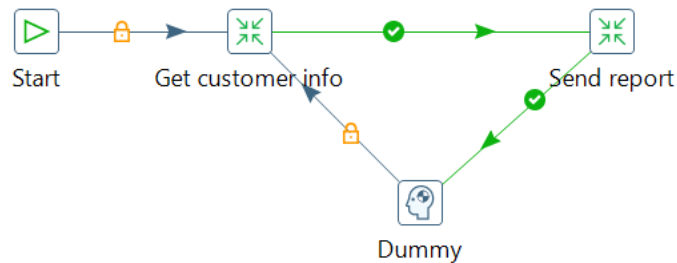


Figure 7: Looping

Rather than looping in the job, set the **Execute every input row** setting on the **Send report** transformation:

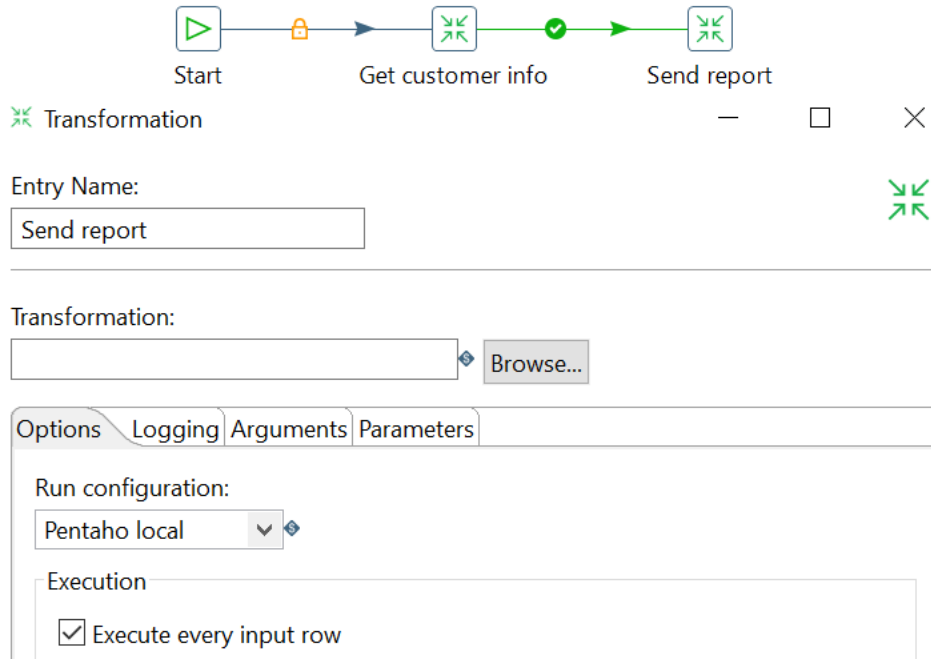


Figure 8: Settings for **Send report** Transformation

The **Get customer 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:** This may be an option if you are using Carte or Pentaho Server. 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 restart jobs from the beginning in case of failure.

## Scaling Up Transformations

This section describes how you can configure transformations and jobs to make the most of your CPU and memory 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, allowing 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 makes use of multiple cores, for example:

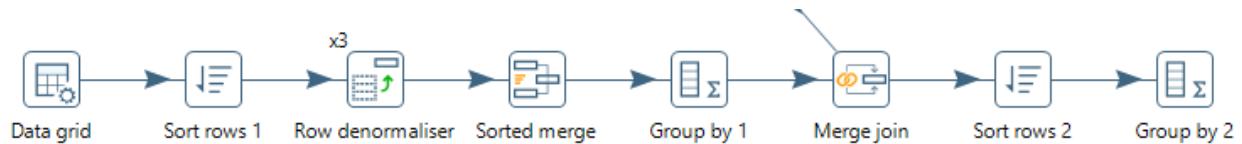


Figure 9: Increasing Number of **Row denormaliser** Step Copies

After **Row denormaliser**, we add a **Sorted merge** because otherwise the data would no longer be sorted at that point in the transformation. The **Row denormaliser** step is assigned three step copies. Each of the copies will spawn its own thread. Therefore, the transformation will spawn a total of three threads that could use up to three cores on the PDI server.



*Make sure not to allocate too many threads, as this can degrade performance. We recommend you keep the number of steps to less than three to four times the number of cores.*

Some other techniques for memory and CPU resources include:

- **Blocking steps:** Both **Blocking step** and **Block this step until steps finish** 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. However, running parallel transformations is preferable to combining transformations, both in terms of code readability and simplicity.

## Memory Utilization

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

### Creating a Row Buffer

Create a row buffer between each step. Since row buffers are stored in memory, this setting allows you to increase or decrease memory used for the buffers. Configure the size of the buffer (in rows) by right-clicking on the transformation, choosing **Settings...**, and going to the **Miscellaneous** tab. There, you can modify the **Nr of rows in rowset** setting.

### Sorting Rows

Sort all rows in memory, since it is significantly faster than using a memory-plus-disk approach. Edit the **Sort rows** step, using the **Sort size (rows in memory)** setting 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 Lookup Steps*

Use joins and lookup steps to configure data caching. The configuration settings control the cache size. This reduces the number of database queries and improves performance, at the cost of using more memory.

## Scaling Out Transformations

While scaling up transformations requires you to add resources to a single server to run a transformation, scaling out allows you to use multiple servers to run a transformation. When you scale out a transformation, each server you use is processing *part* of the transformation, such as a subset of rows.

More information for setting up and using scaling out of transformations can be found at:

- [Pentaho MapReduce](#)
- [Adaptive Execution Layer \(AEL\)](#)

## Other Related Information

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

- [Amazon EC2 Instance Types](#)
- [Amazon S3](#)
- [Logging, Monitoring, and Performance Tuning for Pentaho](#)
- [Pentaho and Amazon Web Services](#)
- [Pentaho Components Reference](#)
- [Pentaho MapReduce](#)
- [Pentaho Performance Tuning](#)
- [Pentaho Table Output Step](#)
- [Adaptive Execution Layer \(AEL\)](#)

## Finalization Checklist

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

Name of the Project: \_\_\_\_\_

Date of the Review: \_\_\_\_\_

Name of the Reviewer: \_\_\_\_\_

Item	Response	Comments
Did you use the Step Metrics tab to identify your bottleneck?	YES_____ NO_____	
Did you use performance monitoring to help eliminate the bottleneck?	YES_____ NO_____	
Did you measure repeatedly to ensure the elimination of the bottleneck?	YES_____ NO_____	
Did you use a Dummy (do nothing) step to verify the location of your bottleneck?	YES_____ NO_____	
Did you identify any possible external performance factors that may affect PDI performance?	YES_____ NO_____	
Did you use the tips for creating jobs and transformations that reduce bottleneck risk?	YES_____ NO_____	