



The Pentaho Performance Tuning Guide



This document supports Pentaho Business Analytics Suite 4.8 GA and Pentaho Data Integration 4.4 GA, documentation revision October 31, 2012.

This document is copyright © 2012 Pentaho Corporation. No part may be reprinted without written permission from Pentaho Corporation. All trademarks are the property of their respective owners.

Help and Support Resources

If you have questions that are not covered in this guide, or if you would like to report errors in the documentation, please contact your Pentaho technical support representative.

Support-related questions should be submitted through the Pentaho Customer Support Portal at <http://support.pentaho.com>.

For information about how to purchase support or enable an additional named support contact, please contact your sales representative, or send an email to sales@pentaho.com.

For information about instructor-led training on the topics covered in this guide, visit <http://www.pentaho.com/training>.

Limits of Liability and Disclaimer of Warranty

The author(s) of this document have used their best efforts in preparing the content and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, express or implied, with regard to these programs or the documentation contained in this book.

The author(s) and Pentaho shall not be liable in the event of incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the programs, associated instructions, and/or claims.

Trademarks

Pentaho (TM) and the Pentaho logo are registered trademarks of Pentaho Corporation. All other trademarks are the property of their respective owners. Trademarked names may appear throughout this document. Rather than list the names and entities that own the trademarks or insert a trademark symbol with each mention of the trademarked name, Pentaho states that it is using the names for editorial purposes only and to the benefit of the trademark owner, with no intention of infringing upon that trademark.

Company Information

Pentaho Corporation
Citadel International, Suite 340
5950 Hazeltine National Drive
Orlando, FL 32822
Phone: +1 407 812-OPEN (6736)
Fax: +1 407 517-4575
<http://www.pentaho.com>

E-mail: communityconnection@pentaho.com

Sales Inquiries: sales@pentaho.com

Documentation Suggestions: documentation@pentaho.com

Sign-up for our newsletter: <http://community.pentaho.com/newsletter/>

Contents

Introduction.....	4
System Requirements.....	5
Pentaho BA Server Performance Tips.....	6
Move Pentaho Managed Data Sources to JNDI.....	6
Manual Cleanup of the /tmp Directory.....	6
Memory Optimization for the Geo Service Plugin.....	6
Switching to a File-Based Solution Repository.....	7
Turning Off Audit Logging.....	7
Using Apache httpd With SSL For Delivering Static Content.....	8
Testing BA Server Scalability.....	10
Pentaho Reporting Performance Tips.....	12
Caching Report Content.....	12
Result Set Caching.....	12
Streamlining Printed Output.....	13
Paginated Exports.....	13
Table Exports.....	14
HTML Exports.....	14
Pentaho Reporting Configuration Files.....	15
Pentaho Data Integration Performance Tips.....	16
Upgrading to the Latest Release.....	16
Pentaho Data Integration Performance Tuning Tips.....	16
Limiting In-Memory Log Output.....	18
HBase Performance in PDI.....	18
Pentaho Analysis (Mondrian) Performance Tips.....	20
Optimizing Your Infrastructure.....	20
Redesigning Your Data Warehouse.....	20
Switching to an Analytic Database.....	20
Query Optimization.....	21
Optimizing Pentaho Analysis.....	21
Mondrian Cache Control.....	21
Partitioning High-Cardinality Dimensions.....	25
Mondrian Log Analysis.....	25
Configuring Pentaho Analyzer for Large Data Warehouses.....	26
Configuring the Mondrian Engine for Large Data Warehouses.....	26
Redesigning Analyzer Reports for Maximum Performance.....	28
Pentaho Analysis Configuration Files.....	28
Pentaho Data Mining (Weka) Performance Tips.....	29
Vertical Resource Scaling.....	30
Horizontal Resource Scaling.....	31
Clustering the Application Server.....	31
Clustering Requirements.....	31
Sharing the Solution Repository.....	31
Installing and Configuring Apache as a Load Balancer.....	32
Tomcat Configuration.....	35
Copying WAR Files to the Nodes.....	36
Starting and Testing the Cluster.....	36
Changing the Java VM Memory Limits.....	37
Increasing Memory Limits on Microsoft Windows with a Graphical Installation.....	37
Increasing Memory Limits on Linux with a Graphical Install.....	37
Increasing Memory Limits with an Archive or Manual Deployment.....	38
Increasing the Memory Limit in Aggregation Designer.....	38
Increasing the Memory Limit in PDI.....	39
Increasing the Memory Limit in Report Designer.....	39
Increasing the Memory Limit in Weka.....	40

Introduction

This guide is designed to help you discover where your BA Server performance bottlenecks are, along with instructions and suggestions on how to address them.

There are many ways to improve the speed and efficiency of Pentaho software documented in this guide. Each applies to a specific situation and **should never be blindly applied**. Some of the performance tweaks herein will remove functionality and in some cases security from your BA Server instance. Others will assign more system resources to the BA Server, which could in turn impact other services running on the same machine.

To put it more plainly: **Performance always comes at the cost of one or more of: functionality, security, or resources.**

The tips and tricks listed in this guide are meant as an initial set of self-service tasks for improving Pentaho Business Analytics performance. There are much more advanced techniques that may improve performance, but require code changes or major surgical changes to Pentaho software, none of which should ever be attempted without qualified assistance. These techniques are not included in this guide for safety reasons. A Pentaho partner or consultant can assist you with more advanced performance improvements, if required.

System Requirements

Before continuing with this guide, you should already have a working and tested Pentaho Business Analytics installation.

There are no operating system or hardware requirements beyond those implicit in specific performance-tuning tips. Requirements are listed on an individual basis for each tip.



Note: It may be possible to use other versions of Pentaho Business Analytics, and other versions of the software mentioned in this guide, such as Tomcat and Apache, but those configurations are untested. If you stray from the tested configuration, be prepared to dynamically modify the instructions in this guide to accommodate your situation, and be warned that your Pentaho support representative may not be able to assist with your unsupported configuration.

Pentaho BA Server Performance Tips

The Pentaho BA Server ships in a condition designed to work well for the majority of customers. However, deployments that drift toward opposite extremes -- very large and very small -- will need to adjust certain settings, and possibly even remove certain unused functionality, in order to achieve the desired performance goals without adding hardware.

Read through the subsections below and decide which ones apply to your scenario.

Move Pentaho Managed Data Sources to JNDI

Most production BI environments have finely-tuned data sources for reporting and analysis. If you haven't done any data warehouse performance-tuning, you may want to consult [Pentaho Analysis \(Mondrian\) Performance Tips](#) on page 20 for basic advice before proceeding.

Pentaho provides a Data Source Wizard in the Pentaho User Console and a data source dialogue in the Pentaho Enterprise Console that enable business users to develop rapid prototype data sources for ad hoc reporting and analysis. This is a great way to get off the ground quickly, but they are "quick and dirty" and not performant. For maximum performance, you should establish your own JNDI data connections at the Web application server level, and tune them for your database.

JNDI data sources can be configured for Pentaho client tools by adding connection details to the `~/ .pentaho/ simple-jndi/default.properties` file on Linux, or the `%userprofile%\ .pentaho\simple-jndi\default.properties` file on Windows. Design Studio requires that connection details be added to `/pentaho-solutions/system/simple-jndi/jdbc.properties` as well.

Manual Cleanup of the /tmp Directory

Every time you generate content on the BA Server, temporary files are created on the local file system in the `/pentaho-solutions/system/tmp/` directory. In some cases, the BA Server may not properly purge that temporary content, leaving behind orphaned artifacts that can slowly build up and reduce performance on the volume that contains the `pentaho-solutions` directory. One way to address this is to mount the `/tmp` directory on a separate volume, thereby siphoning off all disk thrash associated with creating new content. However, you will still have to perform a manual garbage collection procedure on this directory on a regular basis. You can accomplish this via a script that runs through your system scheduler; it should be safe to delete any content files in this directory that are more than a week old.

Memory Optimization for the Geo Service Plugin

The Pentaho Geo Service enables Geo Map visualizations in Analyzer.

If you do not use Analyzer or are sure that you are not using the Geo Service, you can free up approximately 600MB of RAM by removing the Geo Service plugin. Simply shut down the BA Server and delete the `/pentaho/server/biserver-ee/pentaho-solutions/system/pentaho-geo/` directory.

If you are a heavy user of the Geo Service, update the cache setting for `pentaho-geo-municipality` in the `ehcache.xml`. This file can be found in the `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/WEB-INF/classes` directory.

```
<cache
  name="pentaho-geo-municipality"
  maxElementsInMemory="125000"
  eternal="false"
  overflowToDisk="true"
  timeToIdleSeconds="0"
  timeToLiveSeconds="0"
  diskPersistent="false"
  diskExpiryThreadIntervalSeconds="120"
/>
```

The default setting, `maxElementsInMemory="125000"`, is a relatively low number and might need to be increased if you are using the Geo Service. Pentaho suggests a setting between 275000 and 700000 for a heavy usage. When

you increase the `maxElementsInMemory` setting, less memory is available for other resources. To accommodate higher `maxElementsInMemory` settings, increase the maximum memory allocated to the JVM running Pentaho processes. The default setting is 768m.

Switching to a File-Based Solution Repository

The Pentaho BA Server and Pentaho Enterprise Console must be stopped before executing these instructions.

This procedure changes your default database solution repository into a strictly file-based repository. There will no longer be a database mirroring the content in your `pentaho-solutions` directory, and there will be no way of isolating or securing BI content within the BA Server. However, removing the database overhead means that there will be a substantial performance increase in many instances. This procedure does not, in itself, remove security from the BI Platform, but it does remove access control lists from all content.

1. Edit the `/pentaho-solutions/system/pentahoObjects.spring.xml` file.
2. Comment out the current **`ISolutionRepository`** line, and uncomment the similar line above it.

Alternatively, you can switch the value of **`ISolutionRepository`** from **`org.pentaho.platform.repository.solution.dbbased.DbBasedSolutionRepository`** to **`org.pentaho.platform.repository.solution.filebased.FileBasedSolutionRepository`**.

```
<!-- Uncomment the following line to use a filesystem-based repository. -->
<!--Note: does not support ACLs. -->
<bean id="ISolutionRepository"
  class="org.pentaho.platform.repository.solution.
    filebased.FileBasedSolutionRepository" scope="session" />
<!-- Uncomment the following line to use a filesystem/db-based repository with -->
>!-- meta information stored in a db -->
<!-- <bean id="ISolutionRepository" -->
<!--   class="org.pentaho.platform.repository.solution.dbbased.
<!--     DbBasedSolutionRepository" scope="session" /> -->
```

3. Comment out the **`IMetadataDomainRepository`** line, and uncomment the similar line below it.

Alternatively, you can switch the value of **`IMetadataDomainRepository`** from **`org.pentaho.platform.plugin.services.metadata.SecurityAwareMetadataDomainRepository`** to **`org.pentaho.platform.plugin.services.metadata.MetadataDomainRepository`**.

```
<!-- <bean id="IMetadataDomainRepository" -->
<!-- class="org.pentaho.platform.plugin.services.metadata. -->
<!-- SecurityAwareMetadataDomainRepository" scope="singleton"/> -->
<!-- Use this schema factory to disable PMD security -->
<bean id="IMetadataDomainRepository"
  class="org.pentaho.platform.plugin.services.
    metadata.MetadataDomainRepository" scope="singleton"/>
```

4. Save and close the file.

You've now switched over to a file-based solution repository. You can safely restart your BA Server and Pentaho Enterprise Console server.

Turning Off Audit Logging

Your BA Server must be stopped before performing this procedure.

While audit logging can be useful for monitoring BA Server activity and performance, the act of collecting the necessary audit data can introduce significant memory overhead with the solution database. Follow the instructions below to disable audit logging in the BA Server.



Note: Performing this task will disable all audit functions in the BA Server's administration interface.

1. Open the `/pentaho-solutions/system/pentahoObjects-spring.xml` file with a text editor.

2. Locate the following line:

```
<bean id="IAuditEntry"
  class="org.pentaho.platform.engine.services.audit.AuditSQLEntry"
  scope="singleton" />
```

3. Replace that line with this one:

```
<bean id="IAuditEntry" class="org.pentaho.platform.engine.core.audit.NullAuditEntry"
  scope="singleton" />
```

4. Save and close the file

5. Using a database management tool or command line interface, connect to the Pentaho **hibernate** database.

6. Truncate (but do not drop) the following tables:

- PRO_AUDIT
- PRO_AUDIT_TRANSFORM_TRACKER

7. Exit your database utility and restart the BA Server.

If you need to reverse this process later, you can replace the old configuration line shown above, and use the **Initialize From File** audit function in the **Services** section of the **Administration** tab in the Pentaho Enterprise Console to restore the audit log table structure.

Using Apache httpd With SSL For Delivering Static Content

You can use the Apache httpd Web server to handle delivery of static content and facilitation of socket connections, neither of which is done efficiently through Tomcat alone, especially under heavy traffic or when accepting connections from the Internet.

1. Install Apache 2.2.x -- with SSL support -- through your operating system's preferred installation method. For most people, this will be through a package manager. It's also perfectly valid to download and install the reference implementation from <http://www.apache.org>.

It is possible to use Apache 1.3, but you will have to modify the instructions on your own from this point onward.

2. If it has started as a consequence of installing, stop the Apache server or service.

3. Retrieve or create your SSL keys.

If you do not know how to generate self-signed certificates, refer to the OpenSSL documentation. Most production environments have SSL certificates issued by a certificate authority such as Thawte or Verisign.

4. Check to see if you already have the Tomcat Connector installed on your system. You can generally accomplish this by searching your filesystem for **mod_jk**, though you can also search your **httpd.conf** file for **mod_jk**. If it is present, then you only need to be concerned with the Apache httpd configuration details and can skip this step. If it is not there, then the Tomcat Connector module needs to be installed. If you are using Linux or BSD, use your package manager or the Ports system to install **mod_jk**. For all other platforms, visit the <http://www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/>, then click on the directory for your operating system. The module will be either an **.so** (for Linux, BSD, OS X, and Solaris) or **.dll** (for Windows) file. Save it to your Apache modules directory, which is generally C:\Program Files\Apache Group\Apache2\modules\ on Windows, and /usr/lib/apache2/modules/ on Unix-like operating systems, though this can vary depending on your Apache configuration.

5. Edit your **httpd.conf** file with a text editor and add the following text to the end of the file, modifying the paths and filenames as instructed in the comments:



Note: Some operating systems use modular httpd configuration files and have unique methods of including each separate piece into one central file. Ensure that you are not accidentally interfering with an auto-generated mod_jk configuration before you continue. In many cases, some of the configuration example below will have to be cut out (such as the **LoadModule** statement). In some cases (such as with Ubuntu Linux), httpd.conf may be completely empty, in which case you should still be able to add the below lines to it. Replace **example.com** with your hostname or domain name.

```
# Load mod_jk module
# Update this path to match your mod_jk location; Windows users should change
  the .so to .dll
LoadModule    jk_module  /usr/lib/apache/modules/mod_jk.so
# Where to find workers.properties
# Update this path to match your conf directory location
```



```
JkWorkersFile /etc/httpd/conf/workers.properties
# Should mod_jk send SSL information to Tomcat (default is On)
JkExtractSSL On
# What is the indicator for SSL (default is HTTPS)
JkHTTPSIndicator HTTPS
# What is the indicator for SSL session (default is SSL_SESSION_ID)
JkSESSIONIndicator SSL_SESSION_ID
# What is the indicator for client SSL cipher suit (default is SSL_CIPHER)
JkCIPHERIndicator SSL_CIPHER
# What is the indicator for the client SSL certificated (default is SSL_CLIENT_CERT)
JkCERTSIndicator SSL_CLIENT_CERT
# Where to put jk shared memory
# Update this path to match your local state directory or logs directory
JkShmFile      /var/log/httpd/mod_jk.shm
# Where to put jk logs
# Update this path to match your logs directory location (put mod_jk.log next to
access_log)
JkLogFile      /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel     info
# Select the timestamp log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# Send everything for context /examples to worker named worker1 (ajp13)
# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"
# Mount your applications
JkMount /pentaho/* tomcat_pentaho
# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm
<VirtualHost example.com
ServerName example.com
JkMount /pentaho default
JkMount /pentaho/* default
JkMount /sw-style default
JkMount /sw-style/* default
JkMount /pentaho-style default
JkMount /pentaho-style/* default
</VirtualHost>
```

6. In your Apache configuration, ensure that SSL is enabled by uncommenting or adding and modifying the following lines:

```
LoadModule ssl_module modules/mod_ssl.so
Include conf/extra/httpd-ssl.conf
```

7. Save and close the file, then edit `/conf/extra/httpd-ssl.conf` and properly define the locations for your SSL certificate and key:

```
SSLCertificateFile "conf/ssl/mycert.cert"
SSLCertificateKeyFile "conf/ssl/mycert.key"
```

8. Ensure that your SSL engine options contain these entries:

```
SSLOptions +StdEnvVars +ExportCertData
```

9. Add these lines to the end of the **VirtualHost** section:

```
JkMount /pentaho default
JkMount /pentaho/* default
JkMount /sw-style default
JkMount /sw-style/* default
JkMount /pentaho-style default
JkMount /pentaho-style/* default
```

10. Save and close the file, then create a **workers.properties** file in your Apache conf directory. If it already exists, merge it with the example configuration in the next step.
11. Copy the following text into the new **workers.properties** file, changing the location of Tomcat and Java, and the port numbers and IP addresses to match your configuration:



Note: Remove the **workers.tomcat_home** setting if you are using JBoss.

```
workers.tomcat_home=/home/pentaho/pentaho/server/biserver-ee/tomcat/
workers.java_home=/home/pentaho/pentaho/java/
worker.list=tomcat_pentaho
worker.tomcat_pentaho.type=ajp13
```

Apache httpd is now configured to securely and efficiently handle static content for Tomcat. You should now start Tomcat and httpd, then navigate to your domain name or hostname and verify that you can access the Pentaho Web application.

Testing BA Server Scalability

Improper scalability testing can give you the wrong idea about changes you've made to your BA Server instance.

Before testing, ensure that you're reusing sessions, instead of creating successive new sessions. Creating multiple unnecessary sessions causes the BA Server to run out of memory unless the session timeout in web.xml is set extremely low (1 per minute, for instance); the default is 30 minutes.

Logging into the BA Server is resource-intensive: you must authenticate the user; create a bunch of session data; and run all startup action sequences, which usually store data in the user's session. So if, during testing, you simply string together a bunch of URLs and ignore the established session, you'll create a series of 30-minute sessions and almost certainly run out of memory. The correct way to test the server is to mimic a user's actions from a browser. The sections below explain how to do this.

Sessions and URLs

Most stress test tools (Loadrunner, JMeter, etc.) have session/cookie management options to ensure that they behave like a human user. However, if you're creating your own test scripts, you should follow this process:

1. Log into the server
2. Execute a URL that contains the **userid** and **password** parameters (**&userid=joe&password=password** for example)
3. Using the same session, submit other URLs that don't have the userid/password in them.

Use this process for as many users as you need to test with.

To log out of a session, you can use the **http://localhost:8080/pentaho/Logout** URL; this will invalidate the session if you append the userid and password values of the logged-in user. Without passing those parameters (or, alternatively, specifying the session ID or cookie) on the Logout URL, you will create another new session instead of closing an old one.

This means that two back-to-back **wget** commands in Linux will create two different HTTP sessions on the server unless one of the following conditions is met:

1. **-cookies=on** is specified for both wget commands
2. **-save-cookies** is used on the first wget command to save the cookies to a file, and **-load-cookies** is used on the second wget command to load the session state

Memory and sessions

Out of memory errors in the BI Platform can happen because of what your test script is doing, not necessarily because of any weakness in the platform. You can see just how robust the BI Platform is by taking a look at a production server's natural (human user) load. The following URL will show you what each day's maximum and present number of HTTP sessions are: **http://testserver.example.com/pentaho/public/UserService**.

You can see the Java virtual machine memory settings by examining the the options passed to the Tomcat or JBoss start scripts, or by looking at the **CATALINA_OPTS** system variable, if there is one. The **Xms** and **Xmx** options

define the minimum and maximum amount of memory assigned to the application server. The default settings are not particularly high, and even if you've adjusted them, take note of the number of sessions it takes to use up all of the memory. Also take note of the fact that closing sessions after an out of memory error will return the memory to the available pool, proving that there are no memory leaks or zombie sessions inherent in the BI Platform.

Pentaho Reporting Performance Tips

Pentaho Reporting's default configuration makes certain assumptions about system resources and the size, features, and details of reports that may not meet your specific requirements. If you have large inline subreports, or many parameters, you can run into performance bottlenecks. Fortunately, many performance problems can be mitigated through specific engine and report options. Refer to the sections below that apply to your scenario.

Caching Report Content

You can cache the result sets of parameterized reports so that every time you change a parameter during your user session (all caching is on a per-session basis) you don't have to retrieve a new result set. By default, Pentaho Reporting has result set caching turned on, but you may find some advantage in turning it off or changing the cache thresholds and settings.



Note: When you publish a report to the BA Server, you switch cache and engine configurations from the local Report Designer versions of **ehcache.xml** and **classic-engine.properties** to the server's version inside the Pentaho WAR. These configurations may not be the same, so if you have made changes to the result set cache settings locally, you may want to port those changes over to the BA Server as well.

Result Set Caching

When rendered, a parameterized report must account for every dataset required for every parameter. Every time a parameter field changes, every dataset is recalculated, which can negatively impact performance.

You can avoid gratuitous dataset recalculations by caching parameter datasets. This is accomplished through the EHCache framework built into the BA Server. You can configure specific settings for published reports by editing the **ehcache.xml** file in the `/WEB-INF/classes/` directory inside of the `pentaho.war`. The relevant element is:

Anything containing complex objects is not cached (CLOB and BLOB data types); neither are results coming from a scripting dataset, a Java method call, a table data source, an external data source (computed in an action sequence), or a CDA data source. In all of these cases there is either no point in caching because it would be more expensive than recalculating, or because there are not enough hints available in the involved parameters.

```
<!--
  Defines a cache used by the reporting engine to hold small datasets.
  This cache can be configured to have a separate instance for each
  logged in user via the
  global report configuration. This per-user cache is required if role
  or other security and
  filter information is used in ways invisible for the reporting
  engine.
-->
<cache name="report-dataset-cache"
  maxElementsInMemory="50"
  eternal="false"
  overflowToDisk="false"
  timeToIdleSeconds="300"
  timeToLiveSeconds="600"
  diskPersistent="false"
  diskExpiryThreadIntervalSeconds="120"
/>
```

The other side of the coin is that if a cache exists for too long, when the data source is updated it may not reflect in the report output because it's still using old data. So there is a balance between performance and accuracy that you must tune to your needs.

Result Set Cache Options

These **classic-engine.properties** options control result set caching in parameterized reports.

Option	Purpose	Possible Values
org.pentaho.reporting. .platform.plugin.cache. PentahoDataCache.CachableRowLimit	Number of rows in the dataset that will be cached; the higher the number, the larger the cache and the more disk space is used while the cache is active.	Integer; default value is 10000 .

Streamlining Printed Output

Pentaho Reporting's overall performance is chiefly affected by the amount of printed content that it has to generate. The more content you generate, the more time the Reporting engine will take to perform all layout computations.

Large inline subreports are notorious for poor performance. This is because the layouted output of an inline subreport is always stored in memory. The master report's layouting pauses until the subreport is fully generated, then it's inserted into the master report's layout model and subsequently printed. Memory consumption for this layouting model is high because the full layout model is kept in memory until the report is finished. If there is a large amount of content in the subreport, you will run into "out of memory" exceptions.

An inline subreport that consumes the full width of the root-level band should be converted into a banded subreport. Banded subreports are layouted and all output is generated while the subreport is processed. The memory footprint for that is small because only the active band or the active page has to be held in memory.

When images are embedded from remote servers (HTTP/FTP sources), you must ensure that the server produces a **LastModifiedDate** header. The Reporting engine uses that header as part of its caching system, and if it is missing, the remote images will not be cached, forcing the engine to retrieve them every time they're needed.

Caching must be configured properly via a valid ehcache configuration file, which is stored in the Pentaho Web app in the `/WEB-INF/classes/` directory. If caching is disabled or misconfigured, then there will be performance problems when loading reports and resources.

Within Pentaho Reporting there are three output types, each with its own memory and CPU consumption characteristics. Each is listed below with an explanation of how it is optimized.

Paginated Exports

A pageable report generates a stream of pages. Each page has the same height, even if the page is not fully filled with content. When a page is filled, the layouted page will be passed over to the output target to render it in either a Graphics2D or a streaming output (PDF, Plaintext, HTML, etc.) context.

Page break methods

When the content contains a manual pagebreak, the page will be considered full. If the pagebreak is a **before-print** break, then the break will be converted to an **after-break**, the internal report states will be rolled back, and the report processing restarts to regenerate the layout with the new constraints. A similar rollback happens if the current band does not fit on the page. Because of this, you would generally prefer break-before over break-after.

So for large reports, you might consider removing manual page breaks and limiting the width of bands.

Page states

When processing a pageable report, the reporting engine assumes that the report will be run in **interactive mode**, which allows for parameterization control. To make browsing through the pages faster, a number of page states will be stored to allow report end-users to restart output processing at the point in the report where they adjust the parameters.

Reports that are run to fully export all pages usually do not need to store those page states. A series of Report engine settings controls the number and frequency of the page states stored:

- org.pentaho.reporting.engine.classic.core.performance.pagestates.PrimaryPoolSize=20
- org.pentaho.reporting.engine.classic.core.performance.pagestates.SecondaryPoolFrequency=4
- org.pentaho.reporting.engine.classic.core.performance.pagestates.SecondaryPoolSize=100
- org.pentaho.reporting.engine.classic.core.performance.pagestates.TertiaryPoolFrequency=10

The Reporting engine uses three lists to store page states. The default configuration looks as follows:

1. The first 20 states (Pages 1 to 20) are stored in the **primary pool**. All states are stored with strong references and will not be garbage collected.
2. The next 400 states (pages 21 to 421) are stored into the **secondary pool**. Of those, every fourth state is stored with a strong reference and cannot be garbage collected as long as the report processor is open.
3. All subsequent states (pages > 421) are stored in the **tertiary pool** and every tenth state is stored as strong reference.

So for a 2000-page report, a total of about 270 states will be stored with strong references.

In server mode, the settings could be cut down to:

```
org.pentaho.reporting.engine.classic.core.performance.pagestates.PrimaryPoolSize=1
org.pentaho.reporting.engine.classic.core.performance.pagestates.
  SecondaryPoolFrequency=1
org.pentaho.reporting.engine.classic.core.performance.pagestates.SecondaryPoolSize=1
org.pentaho.reporting.engine.classic.core.performance.pagestates.
  TertiaryPoolFrequency=100
```

This reduces the number of states stored for a 2000 page report to 22, thus cutting the memory consumption for the page states to a 1/10th.


 **Note:** In the current version full exports do not generate page states and thus these settings have no effect on such exports. They still affect the interactive mode.

Table Exports

A table export produces tabular output from a fully-laid out display model. A table export cannot handle overlapping elements and therefore has to remove them.

To support layout debugging, the Reporting engine stores a lot of extra information in the layout model. This increases memory consumption but makes it easier to develop Reporting solutions. These Reporting engine debug settings should never be enabled in production environments:

- org.pentaho.reporting.engine.classic.core.modules.output.table.base.ReportCellConflicts
- org.pentaho.reporting.engine.classic.core.modules.output.table.base.VerboseCellMarkers

 **Note:** These settings are **false** by default. Report Designer comes with its own method to detect overlapping elements and does not rely on these settings.

HTML Exports

In HTML exports, there are a few Reporting engine settings that can affect export performance. The first is **CopyExternalImages**:

```
org.pentaho.reporting.engine.classic.core.modules.output.table.html.CopyExternalImages=
true
```

This controls whether images from HTTP/HTTPS or FTP sources are linked from their original source or copied (and possibly re-encoded) into the output directory. The default is **true**; this ensures that reports always have the same image. Set to **false** if the image is dynamically generated, in which case you'd want to display the most recent view.

The **Style** and **ForceBufferedWriting** settings control how stylesheets are produced and whether the generated HTML output will be held in a buffer until the report processing is finished:

```
org.pentaho.reporting.engine.classic.core.modules.output.table.html.
ForceBufferedWriting=true
```

Style information can be stored inline, or in the <head> element of the generated HTML file:

```
org.pentaho.reporting.engine.classic.core.modules.output.table.html.InlineStyles=true
```

Or in an external CSS file:

```
org.pentaho.reporting.engine.classic.core.modules.output.table.html.ExternalStyle=true
```

ForceBufferedWriting should be set to true if a report uses an external CSS file. Browsers request all resources they find in the HTML stream, so if a browser requests a style sheet that has not yet been fully generated, the report cannot display correctly. It is safe to disable buffering if the styles are inline because the browser will not need to fetch an external style sheet in that case. Buffered content will appear slower to the user than non-buffered content because browsers render partial HTML pages while data is still being received from the server. Buffering will delay that rendering until the report is fully processed on the server.

Pentaho Reporting Configuration Files

The following files contain various configuration options for Pentaho Reporting. The options are not particularly self-explanatory and their value limits are not obvious; therefore, you shouldn't change any options in these files unless you are following guidelines from Pentaho documentation or are assisted by a Pentaho support or consulting representative.

File	Purpose
/pentaho/design-tools/report-designer/resources/ report-designer.properties	Contains options for the Report Designer client tool. It does not change any report options.
/pentaho/design-tools/report-designer/resources/ classic-engine.properties	Contains global report rendering options for reports generated locally from Report Designer. Some of these options can be overridden in individual reports.
/tomcat/webapps/pentaho/WEB-INF/classes/ classic-engine.properties	Contains global report rendering options for published reports that are generated on the BA Server. Some of these options can be overridden in individual reports.

Pentaho Data Integration Performance Tips

Below are some tips and tricks for improving PDI performance. Most of them involve streamlining jobs and transformations.


Upgrading to the Latest Release

You can see some substantial performance increases in enterprise repository transactions by upgrading. Pentaho puts a great deal of effort into improving repository efficiency in the latest release.

Refer to the [Business Analytics Upgrade Guide](#) or the [Pentaho Data Integration Upgrade Guide](#). You can obtain all upgrade materials and instructions from the Pentaho InfoCenter.

Pentaho Data Integration Performance Tuning Tips

The tips described here may help you to identify and correct performance-related issues associated with PDI transformations.

Step	Tip	Description
JavaScript	Turn off compatibility mode	<p>Rewriting JavaScript to use a format that is not compatible with previous versions is, in most instances, easy to do and makes scripts easier to work with and to read. By default, old JavaScript programs run in compatibility mode. That means that the step will process like it did in a previous version. You may see a small performance drop because of the overload associated with forcing compatibility. If you want make use of the new architecture, disable compatibility mode and change the code as shown below:</p> <ul style="list-style-type: none"> <code>intField.getInteger() --> intField</code> <code>numberField.getNumber() --> numberField</code> <code>dateField.getDate() --> dateField</code> <code>bigNumberField.getBigNumber() --> bigNumberField</code> and so on... <p>Instead of Java methods, use the built-in library. Notice that the resulting program code is more intuitive. For example :</p> <ul style="list-style-type: none"> checking for null is now: <code>field.isNull() --> field==null</code> Converting string to date: <code>field.Clone().str2dat() --> str2date(field)</code> and so on... <p>If you convert your code as shown above, you may get significant performance benefits.</p> <p> Note: It is no longer possible to modify data in-place using the value methods. This was a design decision to ensure that no data with the wrong type would end up in the output rows of the step. Instead of modifying fields in-place, create new fields using the table at the bottom of the Modified JavaScript transformation.</p>
JavaScript	Combine steps	One large JavaScript step runs faster than three consecutive smaller steps. Combining processes in one larger step helps to reduce overhead.
JavaScript	Avoid the JavaScript step or write a custom plug in	Remember that while JavaScript is the fastest scripting language for Java, it is still a scripting language. If you do the same amount of work in a native step or plugin, you avoid the overhead of the JS scripting engine. This has been known to result in significant performance gains. It is also the primary reason why the Calculator step was created — to avoid the use of JavaScript for simple calculations.

Step	Tip	Description
JavaScript	Create a copy of a field	No JavaScript is required for this; a "Select Values" step does the trick. You can specify the same field twice. Once without a rename, once (or more) with a rename. Another trick is to use $B=NVL(A,A)$ in a Calculator step where B is forced to be a copy of A. An explicit "create copy of field A" function has been added to the Calculator.
JavaScript	Data conversion	Consider performing conversions between data types (dates, numeric data, and so on) in a "Select Values" step. You can do this in the Metadata tab of the step.
JavaScript	Variable creation	If you have variables that can be declared once at the beginning of the transformation, make sure you put them in a separate script and mark that script as a startup script (right click on the script name in the tab). JavaScript object creation is time consuming so if you can avoid creating a new object for every row you are transforming, this will translate to a performance boost for the step.
Not Applicable	Launch several copies of a step	There are two important reasons why launching multiple copies of a step may result in better performance: <ol style="list-style-type: none"> 1. The step uses a lot of CPU resources and you have multiple processor cores in your computer. Example: a JavaScript step 2. Network latencies and launching multiple copies of a step can reduce average latency. If you have a low network latency of say 5ms and you need to do a round trip to the database, the maximum performance you get is 200 (x5) rows per second, even if the database is running smoothly. You can try to reduce the round trips with caching, but if not, you can try to run multiple copies. Example: a database lookup or table output
Not Applicable	Manage thread priorities	This feature that is found in the "Transformation Settings" dialog box under the (Misc tab) improves performance by reducing the locking overhead in certain situations. This feature is enabled by default for new transformations that are created in recent versions, but for older transformations this can be different.
Select Value	If possible, don't remove fields in Select Value	Don't remove fields in Select Value unless you must. It's a CPU-intensive task as the engine needs to reconstruct the complete row. It is almost always faster to add fields to a row rather than delete fields from a row.
Get Variables	Watch your use of Get Variables	May cause bottlenecks if you use it in a high-volume stream (accepting input). To solve the problem, take the "Get Variables" step out of the transformation (right click, detach) then insert it in with a "Join Rows (cart prod)" step. Make sure to specify the main step from which to read in the "Join Rows" step. Set it to the step that originally provided the "Get Variables" step with data.
Not Applicable	Use new text file input	The new "CSV Input" or "Fixed Input" steps provide optimal performance. If you have a fixed width (field/row) input file, you can even read data in parallel. (multiple copies) These new steps have been rewritten using Non-blocking I/O (NIO) features. Typically, the larger the NIO buffer you specify in the step, the better your read performance will be.
Not applicable	When appropriate, use lazy conversion	In instances in which you are reading data from a text file and you write the data back to a text file, use Lazy conversion to speed up the process. The principle behind lazy conversion that it delays data conversion in hopes that it isn't necessary (reading from a file and writing it back comes to mind). Beyond helping with data conversion, lazy conversion also helps to keep the data in "binary" storage form. This, in turn, helps the internal Kettle engine to perform faster data serialization (sort, clustering, and so on). The Lazy Conversion option is available in the "CSV Input" and "Fixed input" text file reading steps.

Step	Tip	Description
Join Rows	Use Join Rows	You need to specify the main step from which to read. This prevents the step from performing any unnecessary spooling to disk. If you are joining with a set of data that can fit into memory, make sure that the cache size (in rows of data) is large enough. This prevents (slow) spooling to disk.
Not Applicable	Review the big picture: database, commit size, row set size and other factors	Consider how the whole environment influences performance. There can be limiting factors in the transformation itself and limiting factors that result from other applications and PDI. Performance depends on your database, your tables, indexes, the JDBC driver, your hardware, speed of the LAN connection to the database, the row size of data and your transformation itself. Test performance using different commit sizes and changing the number of rows in row sets in your transformation settings. Change buffer sizes in your JDBC drivers or database.
Not Applicable	Step Performance Monitoring	You can track the performance of individual steps in a transformation. Step Performance Monitoring is an important tool that allows you identify the slowest step in your transformation.

Limiting In-Memory Log Output

PDI logs data about transformations and jobs according to default parameters that control how many lines are allowed in the log and how long the oldest line should stay in memory before it is released. Obviously the more lines that are recorded and the longer they are kept, the more heap space is consumed by them. If you are experiencing memory shortages or slow performance in your PDI content, you can address the problem by modifying in-memory logging.

In Spoon, the following parameters control logging:

- **KETTLE_MAX_LOG_SIZE_IN_LINES**, which sets the maximum number of log lines that are kept internally by Kettle. Setting this to **0** (the default) forces PDI to keep all rows.
- **KETTLE_MAX_LOG_TIMEOUT_IN_MINUTES**, which represents the maximum age (in minutes) that a log line should be kept internally by PDI. Setting this to **0** (the default) keeps all rows indefinitely.
- **KETTLE_MAX_JOB_TRACKER_SIZE**, which sets the maximum number of job trackers kept in memory. Default value is 1000.
- **KETTLE_MAX_JOB_ENTRIES_LOGGED**, which sets the maximum number of job entry results kept in memory for logging purposes. Default value is 1000.
- **KETTLE_MAX_LOGGING_REGISTRY_SIZE**, which sets the maximum number of logging registry entries kept in memory for logging purposes. Default value is 1000.

The equivalent parameters to the first two variables, which can be set on each KTR or KJB individually using Kitchen or Pan, are:

- maxloglines
- maxlogtimeout

Set these values to the lowest non-zero values that your operations can tolerate. If you are using logging for any purpose, you must balance between tolerable performance and necessary functionality.

HBase Performance in PDI

HBase Output Performance Considerations

The **Configure connection** tab provides a field for setting the size of the write buffer used to transfer data to HBase. A larger buffer consumes more memory (on both the client and server), but results in fewer remote procedure calls. The default (defined in the hbase-default.xml file) is 2MB. When left blank, the buffer is 2MB, **auto flush** is enabled, and **Put** operations are executed immediately. This means that each row will be transmitted to HBase as soon as it arrives at the step. Entering a number (even if it is the same as the default) for the size of the write buffer will disable auto flush and will result in incoming rows only being transferred once the buffer is full.

There is also a checkbox for disabling writing to the **Write Ahead Log (WAL)**. The WAL is used as a lifeline to restore the status quo if the server goes down while data is being inserted. However, the tradeoff for error-recovery is speed.

The **Create/edit mappings** tab has options for creating new tables. In the **HBase table name** field, you can suffix the name of the new table with parameters for specifying what kind of compression to use, and whether or not to use Bloom filters to speed up lookups. The options for compression are: NONE, GZ and LZO; the options for Bloom filters are: NONE, ROW, ROWCOL. If nothing is selected (or only the name of the new table is defined), then the default of NONE is used for both compression and Bloom filters. For example, the following string entered in the HBase table name field specifies that a new table called "NewTable" should be created with GZ compression and ROWCOL Bloom filters:

```
NewTable@GZ@ROWCOL
```



Note: Due to licensing constraints, HBase does not ship with LZO compression libraries; these must be manually installed on each node if you want to use LZO compression.

HBase Input Performance Considerations

Specifying fields in the Configure query tab will result in scans that return just those columns. Since HBase is a sparse column-oriented database, this requires that HBase check to see whether each row contains a specific column. More lookups equate to reduced speed, although the use of Bloom filters (if enabled on the table in question) mitigates this to a certain extent. If, on the other hand, the fields table in the Configure query tab is left blank, it results in a scan that returns rows that contain all columns that exist in each row (not only those that have been defined in the mapping). However, the HBase Input step will only omit those columns that are defined in the mapping being used. Because all columns are returned, HBase does not have to do any lookups. However, if the table in question contains many columns and is dense, then this will result in more data being transferred over the network.

Pentaho Analysis (Mondrian) Performance Tips

This section contains advice and procedures for testing and improving Mondrian performance. There are two facets of Pentaho Analysis performance to consider: **Query speed** and **execution speed**. Query speed is the amount of time it takes to retrieve data from your data warehouse or data mart, and execution speed is the amount of time it takes to manipulate or perform calculations with that data after it has been retrieved. With that in mind, this should be a rough outline of your performance-tuning process:


- Locate the performance problem. Is this with query speed (retrieving a result set) or execution speed (calculations done client-side and in the Mondrian engine)? **Most commonly, the performance problem is in your data structure, not the Analysis engine or client machine.**
- If query speed is slow, you must reconsider your data warehouse design and implementation.
- If your data warehouse is soundly designed, are you using an analytic database to achieve maximum query performance?
- If execution speed is slow, you may need to do some tuning of the Mondrian or Reporting engines.
- If high-cardinality dimensions are unavoidable, you may need to partition them and streamline your schema to support table partitioning.

The sections below explain these points in greater detail.

Optimizing Your Infrastructure

The guidelines and advice in this section are specific to changes that you can make with your in-house infrastructure. None of the performance-tuning tips in this section have specifically to do with modifying Pentaho software. Before you get to the point where you can confidently tune the Analysis engine and Pentaho Analyzer, you must ensure that everything on your side of the equation is properly optimized.

Redesigning Your Data Warehouse

 **Note:** The advice in this section applies to building and optimizing data warehouses in general and is not specific to Analysis. However, since poor data warehouse design is so frequently a significant source of performance loss for Pentaho Analysis, it is listed in this section.

A data warehouse exists to consolidate and partition transactional data into one streamlined, organized, canonical source for reporting and analysis. Some guidelines to follow in data warehouse design are:

- Be open to modifying the original design to meet adjusted requirements from business users (iterative design).
- Remove data that is not actually used by business users.
- Optimize for the right purpose. There are basically two use cases to consider: analysis (slice/dice/pivot) and static reporting. You could also use a data warehouse to cleanse and consolidate transactional data for data mining, but this model would almost certainly be inappropriate for analysis or reporting.
- Avoid creating high-cardinality dimensions (putting too many records into fact tables). High-cardinality dimensions will never perform well.
- If there is a lot of unrelated information in your data warehouse, consider breaking it up into more topic-specific data marts.
- Create indexes for large fact tables.
- Create aggregate tables for frequently-computed views.

Switching to an Analytic Database

Some databases are better than others for data warehouses and standalone data marts. Databases that are designed for query speed -- not insert speed -- are optimal for storing data for analysis. For this reason, such databases are often referred to as **analytic databases**. Examples include Netezza, InfoBright, Teradata, and Greenplum, though Pentaho does not specifically endorse or recommend any specific analytic database.

If you are not currently using an analytic database as your ROLAP data source, and you are experiencing poor query performance, then switching to an analytic database should be among your first considerations for improving Pentaho Analysis performance.

Query Optimization



Note: This section is still in progress. The information below is accurate, but may be insufficient.

Indexing is a major factor in query performance, and is one valid way of solving the high-cardinality dimension problem without redesigning the data warehouse. Have your database administrator review your database configuration and ensure that large dimensions and measures are properly indexed.

Optimizing Pentaho Analysis

Once you've properly tuned your data warehouse, you can move on to tuning your ROLAP schema, the Mondrian engine, and the Analyzer client tool.

Mondrian Cache Control

This section contains instructions for configuring and controlling the cache infrastructure that the Pentaho Analysis engine uses for ROLAP data. This information is useful for properly updating your ROLAP cubes when your data warehouse is refreshed, and for performance-tuning.



Restriction: Most of the advanced cache features explained in this section are for Enterprise Edition deployments only. Within that, most of the Enterprise Edition features of the Analysis engine are only beneficial to large, multi-node ROLAP deployments that are performing poorly.

The Analysis engine does not ship with a segment cache, but it does have the ability to use third-party cache systems. If you've installed Pentaho Analysis Enterprise Edition, then you have a default configuration for the JBoss **Infinispan** distributed cache, though the actual Infinispan software is not included and must be downloaded separately. Infinispan supports a wide variety of sub-configurations and can be adapted to cache in memory, to the disk, to a relational database, or (the default setting) to a distributed cache cluster.

The Infinispan distributed cache is a highly scalable solution that distributes cached data across a self-managed cluster of Mondrian instances. Every Mondrian instance running the Analysis Enterprise Edition plugin on a local network will automatically discover each other using UDP multicast. An arbitrary number of segment data copies are stored across all available nodes. The total size of the cache will be the sum of all of the nodes' capacities, divided by the number of copies to maintain. This is all fully configurable; options are explained later in this section.

Other supported segment cache configurations include, but are not limited to:

- **Memcached**, which uses an established (extant) Memcached infrastructure to cache and share the segment data among Mondrian peers.
- **Pentaho Platform Delegating Cache**, which relies on the Pentaho BA Server to delegate segment data storage to the BA Server's native caching capabilities, thus leveraging the existing caching configuration. Some people may prefer this configuration because it keeps the BA Server and Analysis engine manageable as a single entity.



Note: The Pentaho Platform Delegating Cache is not yet feature-complete. It will be available in a future Business Analytics release, but it is not yet ready for production use.

Segment Cache Architecture



Restriction: The segment cache features explained in this section are for very large ROLAP deployments, and require a Pentaho Analysis Enterprise Edition license.

How the Analysis Engine Uses Memory

Each Mondrian segment cache node, regardless of which configuration it uses, loads the segments required to answer a given query into system memory. This cache space is called the **query cache**, and it is composed of hard Java references to the segment objects. Each individual node must have enough memory space available to answer any given query. This might seem like a big limitation, but Mondrian uses deeply optimized data structures which usually take no more than a few megabytes, even for queries returning thousands of rows.

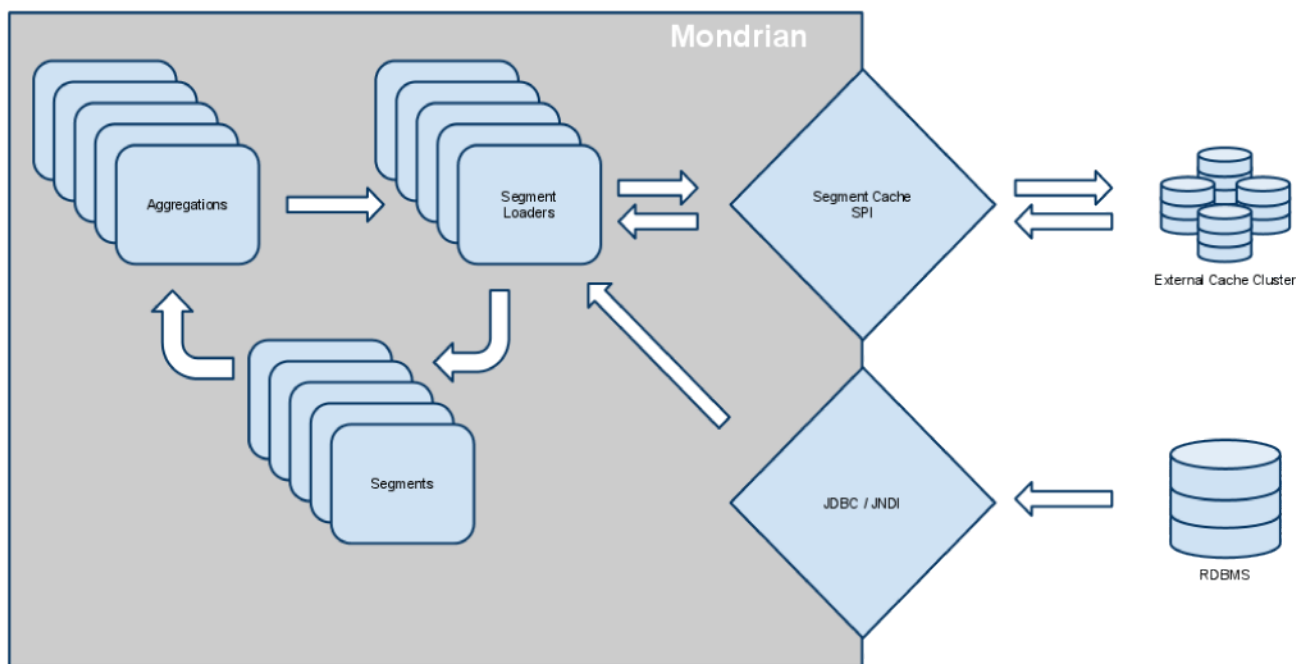
Once the query finishes, Mondrian will usually try to keep the data locally, using a weak reference to the segment data object. A weak reference is a special type of Java object reference which doesn't force the JVM to keep this object in

memory. As the Mondrian node keeps answering queries, the JVM might decide to free up that space for something more important, like answering a particularly big query. This cache is referred to as the **local cache**.

The local cache can be switched on or off by editing the Pentaho Analysis EE configuration file and modifying the value (set it to **true** or **false**) of the **DISABLE_LOCAL_SEGMENT_CACHE** property. Setting this property will not affect the query cache.

This is the order in which Mondrian will try to obtain data for a required segment once a query is received:

1. The node will parse the query and figure out which segments it must load to answer that particular query
2. It checks into the local cache, if enabled.
3. If the data could not be loaded from the local cache, it checks into the external segment cache, provided by the Pentaho Analysis plugin, and it places a copy inside the query cache.
4. If the data is not available from the external cache, it loads the data form SQL and places it into the query cache.
5. If the data was loaded form SQL, it places a copy in the query cache and it sends it to the external cache to be immediately shared with the other Mondrian nodes.
6. The node can now answer the query.
7. Once the query is answered, Mondrian will release the data from the query cache.
8. If the local cache is enabled, a weak reference to the data is kept there.



Cache Control and Propagation

All cache control operations are performed through Mondrian's CacheControl API, which is documented in the Mondrian project documentation at <http://mondrian.pentaho.com>. The CacheControl API allows you to modify the contents of the cache of a particular node. It controls both the data cache and the OLAP schema member cache.

When flushing a segment region on a node, that node will propagate the change to the external cache by using the SegmentCache SPI. If the nodes are not using the local cache space, then the next node to pick up a query requiring that segment data will likely fetch it again through SQL. Once the data is loaded from SQL, it will again be stored in the external segment cache.

You should not use the local cache space when you are using the external cache. For this reason, it is disabled by default in Pentaho Analysis Enterprise Edition.


Using the local cache space on a node can improve performance with increased data locality, but it also means that all the nodes have to be notified of that change. Mondrian nodes don't propagate the cache control operations among the members of a cluster. If you deploy a cluster of Mondrian nodes and don't propagate the change manually across all of them, then some nodes will answer queries with stale data.

Cache Configuration Files


The following files contain configuration settings for Pentaho Analysis cache frameworks. All of them are in the same directory inside of the deployed pentaho.war: `/WEB-INF/classes/`.

- **pentaho-analysis-config.xml** Defines the global behavior of the Pentaho Analysis Enterprise Edition plugin. Settings in this file enable you to define which segment cache configuration to use, and to turn off the segment cache altogether.
- **infinispan-config.xml** The InfinispanSegmentCache settings file. It configures the Infinispan system.
- **jgroups-udp.xml** Configures the cluster backing the Infinispan cache. It defines how the nodes find each other and how they communicate. By default, Pentaho uses UDP and multicast discovery, which enables you to run many instances on a single machine or many instances on many machines. (There are examples of other communication setups included in the JAR archive.) This file is referenced by `infinispan` as specified in the `infinispan-config.xml` configuration file.
- **memcached-config.xml** Configures the Memcached-based segment cache. It is not used by default. To enable it, modify `SEGMENT_CACHE_IMPL` in `pentaho-analysis-config.xml`.

Modifying the JGroups Configuration

 **Restriction:** The segment cache features explained in this section are for very large ROLAP deployments, and require a Pentaho Analysis Enterprise Edition license.

The default Infinispan configuration uses JGroups to distribute the cache across all Mondrian instances it finds on the local network. If you want to modify how those communications are done, you must edit the JGroups configuration file.


 **Note:** Fine-grained JGroups configuration is covered in the JGroups documentation; you should read through it before making changes.

Each node might require a different configuration, so although the default configuration is highly portable, it might not work for you.

If you are deploying this plugin on Amazon EC2, JGroups has a special configuration file that you copied to your `/WEB-INF/classes/` directory when you installed the Analysis Enterprise Edition package. Additionally, default JGroups configuration files are inside of the JAR archive. To switch implementations, edit `infinispan-config.xml` and make the modification appropriate to your communication method:

Comm. type	Config entry
UDP communication	<pre><property name="configurationFile" value="jgroups-udp.xml" /></pre>
TCP communication	<pre><property name="configurationFile" value="jgroups-tcp.xml" /></pre>
Amazon EC2	<pre><property name="configurationFile" value="jgroups-ec2.xml" /></pre>

Switching to Another Cache Framework

 **Restriction:** The segment cache features explained in this section are for very large ROLAP deployments, and require a Pentaho Analysis Enterprise Edition license.

Pentaho Analysis Enterprise Edition ships with configuration files that assume a JBoss Infinispan deployment. Instructions are provided below for switching to the Pentaho Platform Delegating Cache or Memcached. However, **Pentaho strongly recommends Infinispan over Memcached for maximum ROLAP performance.**

Also in this section is a brief overview of how to create a Java class to implement your own custom cache system.

Switching to Memcached

In order to complete this procedure, you must have your own pre-configured Memcached instance. You should have also installed the Analysis Enterprise Edition package to your BA Server or standalone Mondrian engine.

If you already use the Memcached cache framework in your organization and would like to hook it up to the Pentaho Analysis ROLAP engine, follow the directions below to switch from the default Infinispan cache framework configuration.



Caution: Pentaho and Mondrian developers recommend against using Memcached. You are almost certain to have better performance with Infinispan.

1. If the BA Server or standalone Mondrian engine are running, shut them down now.
2. If you performed a default install of the Pentaho Analysis Enterprise Edition package, then you should have all of the required JARs installed to the BA or Mondrian server. If you aren't sure, verify now that the following JARs are present in the `/WEB-INF/lib/` directory inside of your deployed pentaho.war or Mondrian engine:
 - pentaho-analysis-ee
 - commons-lang
 - commons-io
 - commons-codec
 - pentaho-ee-dsc-core
 - memcached
3. Edit the **pentaho-analysis-config.xml** in the `/WEB-INF/classes/` directory inside the deployed pentaho.war or Mondrian engine, and change the value of **SEGMENT_CACHE_IMPL** to match the class name referenced below:

```
<entry key="SEGMENT_CACHE_IMPL">com.pentaho.analysis.segmentcache.impl.memcached.MemcachedSegmentCache</entry>
```

4. Edit the **memcached-config.xml** in the `/WEB-INF/classes/` directory inside the deployed pentaho.war or Mondrian engine, and change the values of **SALT**, **SERVERS**, and **WEIGHT** to match your preference:

```
<entry key="SALT">YOUR SECRET SALT VALUE HERE</entry>
<entry key="SERVERS">192.168.0.1:1642,192.168.0.2:1642</entry>
<entry key="WEIGHTS">1,1</entry>
```

Your Pentaho Analysis Enterprise Edition instance is now configured to use Memcached for ROLAP segment caching. [Memcached Configuration Options](#)

These properties control Memcached settings, and are set in the **memcached-config.xml** file in the `/WEB-INF/classes/` directory inside of your deployed pentaho.war or Mondrian engine.



Note: This is not a comprehensive list of the potential Memcached settings; the options explained below are the ones most critical to Memcached configuration for Pentaho Analysis.

Property	Purpose
SERVERS	A comma-separated list of servers and port numbers representing the Memcached nodes usable by the plugin.
WEIGHTS	A comma-separated list of numbers representing the relative caching capacity of the servers defined in the SERVERS property. There must be exactly as many values of WEIGHTS as there are values of SERVERS. As an example, if the first server has a capacity of 128 megabytes, and the second has a capacity of 256 megabytes, the correct values for the WEIGHTS property should be "1,2", indicating that the first server has a relative size of half of the second one.
SALT	A secret key prefix to be used when saving and loading segment data from the Memcached nodes. This property must be the same for all Mondrian nodes that share their caches. If the SALT value is different from one node to the next, the nodes will not be able to share their cache data.

Switching to Pentaho Platform Delegating Cache

In order to complete this procedure, you must have installed the Analysis Enterprise Edition package to your BA Server.

If you would like to share the BA Server solution cache with the Pentaho Analysis segment cache, follow the directions below.

This cache system is still experimental and not fully implemented; it is not recommended for production use. Therefore, no public documentation is available at this time.

Using a Custom SegmentCache SPI

If you want to develop your own implementation of the SegmentCache SPI, you'll have to follow this basic plan:

1. Create a Java class that implements **mondrian.spi.SegmentCache**
2. Compile your class and make it available in Mondrian's classpath
3. Edit **mondrian.properties** and set **mondrian.rolap.SegmentCache** to your class name
4. Start the BA Server or Mondrian engine

This is only a high-level overview. If you need more specific advice, contact your Pentaho support representative and inquire about developer assistance.

Clearing the Mondrian Cache

There is a default action sequence in the BA Server that will clear the Mondrian cache, which will force the cache to rebuild when a ROLAP schema is next accessed by the BA Server.

The cache-clearing action sequence is **clear_mondrian_schema_cache.xaction**, and you can find it in the **admin** solution directory. This action sequence can be run directly from a URL by making the **admin** solution directory visible and then running the action sequence from the solution browser in the Pentaho User Console, from the Pentaho User Console by selecting the **Mondrian Schema Cache** entry in the **Refresh** part of the **Tools** menu, or by clicking the **Mondrian Cache** button in the **Administration** section of the Pentaho Enterprise Console:

```
http://localhost:8080/admin/clear_mondrian_schema_cache.xaction
```

Partitioning High-Cardinality Dimensions

If you cannot avoid creating high-cardinality dimensions, then you must devise a strategy to make them more performant without reducing their size. Typically a database will partition large tables, which makes querying one partition a quick operation. However, the Analysis engine does not have any way of detecting which tables are partitioned and which are not. Therefore, MDX queries will be translated into SQL statements that are too broad, resulting in a query that traverses all of a table's partitions.

To instruct the Analysis engine to properly address a (partitioned) high-cardinality dimension, you must modify the ROLAP schema and explicitly set the **highCardinality** property of the **ElementCubeDimension** element to **true** on each applicable dimension. This will streamline SQL generation for partitioned tables; ultimately, only the relevant partitions will be queried, which could greatly increase query performance.

Mondrian Log Analysis

In order to determine the causes of your Analysis performance problems, you must enable logging in the Analysis engine and your data warehouse database so that you can view information about the infrastructure, and both the SQL and MDX queries involved in your Analysis calculations. Your DBA should perform the initial database performance-tuning work by looking at the database logs, making sure statistics are up to date (access plans are computed and rational), and your usage is profiled. Make sure the aggregation levels are based on the top 50-80 common uses.

Base all of your performance tuning on this data; it will tell you everything you need to know about bottlenecks in your data structure.

You can also determine the causes behind hanging queries in an Analyzer report by viewing Mondrian log information directly through the Analyzer interface:

1. Log into the BA Server as an administrator
2. Create or load an Analyzer report
3. Click the **Clear Cache** link near the top of the report
4. Click the **XML** link to the left of Clear Cache, and then click **OK**
5. Click the **Log** link between XML and Clear Cache

A new browser tab will open with log information about the open report. You can refresh this page to see the query progress in real time. The following log entries are the most important to watch out for:

- If each SQL query is reported twice. The first time is for Mondrian to get the first record and the second time is to retrieve all records
- SQL queries with high execution times
- SQL queries that return large volumes of data (more than 1000 rows)
- SQL queries that don't join tables
- SQL queries that don't include filters
- This log entry: **WARN mondrian.rolap.RolapUtil Unable to use native SQL evaluation for 'NonEmptyCrossJoin'; reason: arguments not supported.** If you see this, try switching the **contains** filter into an **includes** filter, or make the contains filter more selective

Configuring Pentaho Analyzer for Large Data Warehouses

Analyzer has some low-level configuration options that will improve performance when working with large data warehouses and high-cardinality dimensions:

- `filter.members.max.count=500`
- `filter.dialog.apply.report.context=false`
- `filter.dialog.useTopCount=true`
- `report.request.service.result.expire.time.seconds=30`
- `report.request.service.result.cleanup.time.seconds=300`

These **analyzer.properties** settings are explained in more detail below.

filter.members.max.count

Controls the maximum number of values to show in the filter dialogue, such as include/exclude filters and date range dropdowns.

filter.dialog.apply.report.context

If set to **true**, when showing available members in the filter dialog, Analyzer will limit those members to the existing filters or measures on the report. This means that when retrieving the list of members, Analyzer will perform the join in the fact table and then apply dimension filters. For a high-cardinality dimension, this may significantly reduce the list of members loaded into memory.

filter.dialog.useTopCount

If both this and **mondrian.native.topcount.enable** in **mondrian.properties** are set to **true**, when showing the first set of members in the filter dialogue, Analyzer will only show that set of members sorted within hierarchy. For high-cardinality dimensions, this is required to avoid loading all members into memory. However, if a user uses the **Find** box in the filter dialogue or if you have **filter.dialog.apply.report.context** set to **true**, then the TopCount will not be used.

report.request.service.result.expire.time.seconds

Report results are released after this amount of time has passed.

Analyzer report requests are processed asynchronously and immediately cleaned up after the first download. While this is efficient because clients usually don't need to download a report more than once, it causes issues with popup blockers that will block the first download and re-submit the download after prompting the user. If you expire the request after 30 seconds, you will work around the popup blocker issues while also enabling people to refresh the browser to re-download a report. This only applies to PDF, Excel or CSV downloads.

report.request.service.result.cleanup.time.seconds

Report result cleanup occurs after this amount of time.

Configuring the Mondrian Engine for Large Data Warehouses

There are several **mondrian.properties** options that control how the Analysis engine interacts with large data warehouse volumes in conjunction with Pentaho Analyzer:

- `mondrian.result.limit=5000000`
- `mondrian.rolap.iterationLimit=5000000`

- `mondrian.rolap.queryTimeout=300`
- `mondrian.native.crossjoin.enable=true`
- `mondrian.native.topcount.enable=true`
- `mondrian.native.filter.enable=true`
- `mondrian.native.nonempty.enable=true`
- `mondrian.rolap.maxConstraints=1000`
- `mondrian.native.ExpandNonNative=true`
- `mondrian.native.EnableNativeRegexpFilter=true`
- `mondrian.expCache.enable=true`

Below are explanations for each property.

mondrian.result.limit

Controls the largest cross join size that Mondrian will handle in-memory. Ideally, no queries should involve large cross joins in-memory; instead, they should be handled by the database.

mondrian.rolap.iterationLimit

This is similar to **mondrian.result.limit**, except this applies to calculating aggregates in-memory such as SUM, MAX, AGGREGATE, etc. This should be set to the same value as `mondrian.result.limit`.

mondrian.rolap.queryTimeout

If any query runs past this number of seconds, then the query is immediately cancelled. The total sum of all SQL statements to process a single MDX statement must be less than this timeout. Setting this to zero disables query timeout, which is not recommended because runaway queries can deprive system resources from other necessary processes.

mondrian.native.crossjoin.enable

If this is set to **true**, when Mondrian needs to cross join multiple dimensions in a report, if the cross join is non-empty (doesn't have a fact relationship), then the join operation will be done via SQL. The resultant SQL query will only return combined dimension members that actually have fact data. This will typically reduce the amount of tuples that need to be processed, and is critical for performance on high-cardinality dimensions.

mondrian.native.topcount.enable

If set to **true**, when fetching the first set of records for the filter dialog, Mondrian will only read that set of records into memory. If set to **false**, all records from the dimension level will be read into memory.

mondrian.native.nonempty.enable

If set to **true**, Mondrian will validate each member in the MDX via SQL. If set to **false**, Mondrian will traverse from parent to child tokens in the member. For high-cardinality dimensions, this must be enabled to avoid reading all members into cache.

mondrian.rolap.maxConstraints

This should be set to the largest number of values that the data warehouse database supports in an IN list.

mondrian.native.ExpandNonNative

Works in conjunction with native evaluation of cross joins. If set to **true**, Mondrian will expand cross join inputs to simple member lists that are candidates for pushdown.

mondrian.native.EnableNativeRegexpFilter

When evaluating a Filter MDX function with a regular expression predicate, if this property is set to **true**, and if the RDBMS dialect supports regular expressions, the Mondrian engine will try to pass down the regular expression to the underlying RDBMS and perform a native filter evaluation.

Redesigning Analyzer Reports for Maximum Performance

Once you have an idea of what you want to show with your Analyzer report, you will almost certainly have to redesign it to be more performant. Because an Analyzer report is basically a hierarchical list of actions, the order in which fields and filters are added to the report can make a big difference in query response time. Even though this does not change the report's graphical output, what happens behind the scenes can make that output display more quickly.

When you re-create your reports, follow this process for best performance:

1. Add and filter by low-cardinality dimensions first
2. Add measures to the report
3. Add high-cardinality dimensions last



Note: When filtering, always choose include/exclude over contains/doesn't contain.

Pentaho Analysis Configuration Files

The following files contain various configuration options for Pentaho Analysis. The options are not particularly self-explanatory and their value limits are not obvious; therefore, you shouldn't change any options in these files unless you are following guidelines from Pentaho documentation or are assisted by a Pentaho support or consulting representative.

File	Purpose
/pentaho/server/biserver-ee/pentaho-solutions/system/mondrian/ mondrian.properties	Contains low-level configuration options for the Pentaho Analysis (Mondrian) engine.
/pentaho/server/biserver-ee/pentaho-solutions/system/analyzer/ analyzer.properties	Contains low-level configuration options for Pentaho Analyzer. These are not options that can be set through Analyzer's graphical user interface.

Pentaho Data Mining (Weka) Performance Tips

The most common Weka performance issue is the **OutOfMemory** exception. This is caused by using resource-intensive algorithms with large data sources. To address this, refer to [Increasing the Memory Limit in Weka](#) on page 40.

Learning algorithms convert multi-valued discrete fields to binary indicator fields, thus potentially expanding the total number of fields. This sort of pre-processing can result in two copies of the data being held in main memory briefly until the transformation is complete. So even if you have enough memory to complete the task, it could take a while to perform. For this reason, you may need to run Weka on very fast multi-core, multi-CPU 64-bit machines if you are concerned with poor performance.

Beyond this, data mining tuning involves looking at each algorithm you're using and tweaking its parameters to improve the speed and accuracy of the results. This is always data- and algorithm-specific, and requires empirical experimentation. If you are running out of memory or experiencing poor performance, you might consider switching to an incrementally learning algorithm such as:

- Naive Bayes
- Naive Bayes multinomial
- DMNBtext
- AODE and AODEsr
- SPegasos
- SGD
- IB1, IBk and KStar
- Locally weighted learning
- RacedIncrementalLogitBoost
- Cobweb

See this page on the Pentaho Wiki for more details: <http://wiki.pentaho.com/display/DATAMINING/Handling+Large+Data+Sets+with+Weka>.

Vertical Resource Scaling


If your hardware has reached the limits of its performance capabilities, there are a few different approaches to expanding it without adding new machines. The cheapest solution is to switch from a 32-bit environment to a 64-bit one, which can often be accomplished without buying any new hardware. If you've gone as far as you can go with software and need to scale up, there are some key upgrade points to consider for business intelligence deployments.



Note: You may need to adjust your support entitlement contract to complete some of the upgrades described here. The Pentaho BA Server is licensed per CPU, so increasing the number of CPUs in your deployment will require a higher level of service. Contact your Pentaho sales representative for more details on support plan pricing.

Horizontal Resource Scaling

If you've gone as far as you can (or care to) go with vertical scalability, then it's time to consider scaling horizontally. The Pentaho BA Server can scale out to a cluster, or further to a cloud environment. Clusters are excellent for permanently expanding resources commensurate with increasing load; cloud computing is particularly useful if you only need to scale out for specific periods of increased activity.


 **Note:** You may need to adjust your support entitlement contract to complete some of the upgrades described here. Contact your Pentaho sales representative for more details on support plan pricing.

Clustering the Application Server

The Pentaho BA Server is scalable out to the limits of its Java application server. Most application servers are easily clustered and load balanced by using the Apache httpd Web server with the Tomcat Connector (**mod_jk**) plugin.

The Tomcat Connector module forwards requests from httpd to Tomcat via the AJP protocol. It can operate locally through localhost or remotely through TCP. Because JBoss embeds Tomcat, the core methodology for clustering between the two application servers is the same:

1. Establish a load balancer that runs Apache httpd with the Tomcat Connector (mod_jk) plugin
2. Copy the pentaho-solutions directory to a network drive; this will be shared among all nodes
3. Configure the application server nodes with identical configurations and Pentaho BA Server deployments

 **Note:** You may have to modify these instructions to accommodate your operating system's unique httpd, Apache module, and application server configuration. Many operating systems -- especially Linux distributions -- have non-standard ways of managing these services.

Clustering Requirements

In order to successfully implement a Pentaho deployment on a Tomcat or JBoss cluster, you must meet the following requirements:

- **Each node and the load balancer must be time-synchronized via NTP.** All machines that comprise the cluster have to have the same system time. If they do not, it will be impossible to share sessions among nodes.
- **You must run only one node per machine (or NIC).** It is possible to run multiple application servers on each node with a modified configuration, but this scenario does not offer any benefit for load balancing (performance) or hardware failover (redundancy), and therefore not covered in this guide. Refer to your application server's clustering documentation for more information.
- **You must use the Tomcat or JBoss version described in the *Compatibility Matrix: Supported Components* section of the installation guides.** You may be able to use this guide as a basic blueprint for configuring other application servers or versions of Tomcat and JBoss for a clustered environment, but Pentaho only supports the versions in the *Compatibility Matrix*.
- **You must have permission to install software and modify service configurations.** Or you must have access to someone at your company who does have the correct permission levels (root access, typically).
- **Only the Pentaho BA Server will be deployed to the cluster.** It is possible to modify the configuration to deploy other WARs or EARs. However, for ease of testing and support, Pentaho only supports deployment of the **pentaho** and **pentaho-style** WARs to the cluster.
- **You must use a single repository location.** Most people use a database-based solution repository; remember that you are not clustering the database server in this procedure -- only the application server. If you are using a file-based repository, you will have to create one canonical location for the solution repository, preferably on a network share so that the location can be the same for all nodes.

Sharing the Solution Repository

In order to keep all cluster nodes current with stored Pentaho content and settings, they must share the pentaho-solutions directory. This directory is mirrored in the Pentaho solution database, which is also shared among all nodes. Since the database configuration is partially stored in this same directory, this is a required step for sharing the BA Server configuration among all nodes as well. Follow the directions below to set up a common pentaho-solutions directory.

1. Select a secure location in your computing infrastructure for the pentaho-solutions directory, and configure it so that it can securely share directories over your corporate network.
2. If you have existing content and BA Server settings that you want to move to the cluster, copy it over to the shared drive now. You can also copy an extant pentaho-solutions directory from a standalone BA Server that you are moving to a clustered environment.
3. Configure each cluster node to automatically connect to the machine that will serve the pentaho-solutions directory. This is best accomplished by setting up a network drive that connects to the remote machine.
4. Modify each BA Server node configuration to point to the new location of the pentaho-solutions directory by editing the `/WEB-INF/web.xml` inside of the deployed **pentaho.war** or unpacked `/webapps/pentaho/` directory and modifying the **solution-path** element:

```
<context-param>
  <param-name>solution-path</param-name>
  <param-value>/mnt/sharedhost/pentaho-solutions/</param-value>
</context-param>
```


5. If you have been using a local solution database (on the same machine with a standalone BA Server that you are migrating to a cluster), you must either move the database to a machine that can be reliably shared among all nodes, or ensure that it is available to the other nodes. If you move the database, you must change the **hibernate** and **quartz** locations in the following files within the shared pentaho-solutions directory:
 - `/system/applicationContext-spring-security-hibernate.properties`
 - `/system/hibernate/hibernate-settings.xml`
 - One of these, depending on which database you are using:
 - `/system/hibernate/mysql.hibernate.cfg.xml`
 - `/system/hibernate/postgresql.hibernate.cfg.xml`
 - `/system/hibernate/oracle.hibernate.cfg.xml`

And **/META-INF/context.xml** inside the pentaho.war or unpacked pentaho directory on each Tomcat or JBoss node.

You now have a pentaho-solutions directory and solution database shared among all of your cluster nodes.

Installing and Configuring Apache as a Load Balancer

Clustering requires a load balancer to manage each application server node. This is most easily accomplished via the Apache httpd Web server, which will connect to each application server node via the Tomcat Connector module.

 **Note:** If you've already installed Apache httpd to serve static Web content for Tomcat, then some of the instructions below will be redundant.

1. Install Apache 2.2.x -- with SSL support -- through your operating system's preferred installation method. For most people, this will be through a package manager. It's also perfectly valid to download and install the reference implementation from <http://www.apache.org>.
It is possible to use Apache 1.3, but you will have to modify the instructions on your own from this point onward.
2. If it has started as a consequence of installing, stop the Apache server or service.
3. Retrieve or create your SSL keys.
If you do not know how to generate self-signed certificates, refer to the OpenSSL documentation. Most production environments have SSL certificates issued by a certificate authority such as Thawte or Verisign.
4. Check to see if you already have the Tomcat Connector installed on your system. You can generally accomplish this by searching your filesystem for **mod_jk**, though you can also search your **http.conf** file for **mod_jk**. If it is present, then you only need to be concerned with the Apache httpd configuration details and can skip this step. If it is not there, then the Tomcat Connector module needs to be installed. If you are using Linux or BSD, use your package manager or the Ports system to install **mod_jk**. For all other platforms, visit the <http://www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/>, then click on the directory for your operating system. The module will be either an **.so** (for Linux, BSD, OS X, and Solaris) or **.dll** (for Windows) file. Save it to your Apache modules directory, which is generally `C:\Program Files\Apache Group\Apache2\modules\` on Windows, and `/usr/lib/apache2/modules/` on Unix-like operating systems, though this can vary depending on your Apache configuration.
5. Edit your **httpd.conf** file with a text editor and add the following text to the end of the file, modifying the paths and filenames as instructed in the comments:



Note: Some operating systems use modular httpd configuration files and have unique methods of including each separate piece into one canonical file. Ensure that you are not accidentally interfering with an auto-generated mod_jk configuration before you continue. In many cases, some of the configuration example below will have to be cut out (such as the **LoadModule** statement). In some cases (such as with Ubuntu Linux), httpd.conf may be completely empty, in which case you should still be able to add the below lines to it. Replace **example.com** with your hostname or domain name.

```
# Load mod_jk module
# Update this path to match your mod_jk location; Windows users should change
  the .so to .dll
LoadModule      jk_module  /usr/lib/apache/modules/mod_jk.so
# Where to find workers.properties
# Update this path to match your conf directory location
JkWorkersFile  /etc/httpd/conf/workers.properties
# Should mod_jk send SSL information to Tomcat (default is On)
JkExtractSSL  On
# What is the indicator for SSL (default is HTTPS)
JkHTTPSIndicator  HTTPS
# What is the indicator for SSL session (default is SSL_SESSION_ID)
JkSESSIONIndicator  SSL_SESSION_ID
# What is the indicator for client SSL cipher suit (default is SSL_CIPHER)
JkCIPHERIndicator  SSL_CIPHER
# What is the indicator for the client SSL certificated (default is SSL_CLIENT_CERT)
JkCERTSIndicator  SSL_CLIENT_CERT
# Where to put jk shared memory
# Update this path to match your local state directory or logs directory
JkShmFile      /var/log/httpd/mod_jk.shm
# Where to put jk logs
# Update this path to match your logs directory location (put mod_jk.log next to
  access_log)
JkLogFile      /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel     info
# Select the timestamp log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# Send everything for context /examples to worker named worker1 (ajp13)
# JkOptions indicates to send SSK KEY SIZE
JkOptions      +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"
# Mount your applications on the load balancer node
JkMount /pentaho/* loadbalancer
# There should be no need to cluster the style WAR, but just in case...
#JkMount /pentaho-style/* loadbalancer
# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm
# Add jkstatus for managing runtime data
<Location /jkstatus/>
JkMount status
Order deny,allow
Deny from all
Allow from 127.0.0.1
</Location>
<VirtualHost example.com
  ServerName example.com
  JkMount /pentaho default
  JkMount /pentaho/* default
  JkMount /sw-style default
  JkMount /sw-style/* default
  JkMount /pentaho-style default
  JkMount /pentaho-style/* default
</VirtualHost>
```

6. In your Apache configuration, ensure that SSL is enabled by uncommenting or adding and modifying the following lines:

```
LoadModule ssl_module modules/mod_ssl.so
Include conf/extra/httpd-ssl.conf
```

7. Save and close the file, then edit `/conf/extra/httpd-ssl.conf` and properly define the locations for your SSL certificate and key:

```
SSLCertificateFile "conf/ssl/mycert.cert"
SSLCertificateKeyFile "conf/ssl/mycert.key"
```

8. Ensure that your SSL engine options contain these entries:

```
SSLOptions +StdEnvVars +ExportCertData
```

9. Add these lines to the end of the **VirtualHost** section:

```
JkMount /pentaho default
JkMount /pentaho/* default
JkMount /sw-style default
JkMount /sw-style/* default
JkMount /pentaho-style default
JkMount /pentaho-style/* default
```

10. Save and close the file, then create a **workers.properties** file in your Apache conf directory. If it already exists, merge it with the example configuration in the next step.
11. Copy the following text into the new **workers.properties** file, changing the location of Tomcat and Java, and the port numbers and IP addresses to match your configuration:



Note: Remove the **workers.tomcat_home** setting if you are using JBoss.

```
# Load-balancer settings
workers.tomcat_home=/home/pentaho/pentaho/server/biserver-ee/tomcat/
workers.java_home=/home/pentaho/pentaho/java/
# Define list of workers that will be used for mapping requests
worker.list=loadbalancer,status
# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=192.168.3.6
worker.node1.type=ajp13
worker.node1.lbfactor=1
worker.node1.cachesize=50
# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host=192.168.3.7
worker.node2.type=ajp13
worker.node2.lbfactor=1
worker.node2.cachesize=50
# Load-balancing behaviour
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1
# Status worker for managing load balancer
worker.status.type=status
```

Apache httpd is now configured to act as a load balancer for two nodes. You can come back and adjust this for more nodes later, but it is easier to test and adjust your configuration at this minimal level before you expand out further.

Proceed to the JBoss or Tomcat section below, depending on which application server you are using.

Tomcat Configuration

Before continuing, you should have a working BA Server instance running in Tomcat 6. This should not be currently in production. The safest way to proceed -- and the quickest way to recover from a failed deployment -- is to make a backup archive of the **tomcat** directory before making any changes.

Follow the directions below to modify your Tomcat server to act as a member of a cluster.

1. Stop Tomcat on each cluster node.
2. Edit the `/tomcat/conf/server.xml` file on each node and add the **jvmRoute** parameter (change the value to match the node names you defined in **workers.properties**) to the **Engine** element:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node01">
```

3. Further down in the file, ensure that the AJP connector line is uncommented, and that the port number matches the node definition you defined in **workers.properties** (if it doesn't match, change the node entry in **workers.properties**, not the AJP connector entry):

```
<Connector URIEncoding="UTF-8" port="8009" enableLookups="false" redirectPort="8443" protocol="AJP/1.3" />
```

4. Also ensure that the SSL version of the AJP connector line is uncommented, and add two properties at the end of the line that define your SSL keystore password:

```
<Connector URIEncoding="UTF-8" port="8443" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" keystorePass="password" />
```

5. Further down in `server.xml`, uncomment the **Cluster** node, but do not make any changes to it:

```
<Cluster className="org.apache.catalina.cluster.tcp.SimpleTcpCluster"
managerClassName="org.apache.catalina.cluster.session.DeltaManager"
expireSessionsOnShutdown="false"
useDirtyFlag="true"
notifyListenersOnReplication="true">

  <Membership
className="org.apache.catalina.cluster.mcast.McastService"
mcastAddr="228.0.0.4"
mcastPort="45564"
mcastFrequency="500"
mcastDropTime="3000"/>

  <Receiver
className="org.apache.catalina.cluster.tcp.ReplicationListener"
tcpListenAddress="auto"
tcpListenPort="4001"
tcpSelectorTimeout="100"
tcpThreadCount="6"/>

  <Sender
className="org.apache.catalina.cluster.tcp.ReplicationTransmitter"
replicationMode="pooled"
ackTimeout="15000"
waitForAck="true"/>

  <Valve className="org.apache.catalina.cluster.tcp.ReplicationValve"
filter=".*\.(gif|.*\.(js|.*\.(jpg|.*\.(png|.*\.(htm|.*\.(html|.*
\.css|.*\.(txt|.*\.(

  <Deployer className="org.apache.catalina.cluster.deploy.FarmWarDeployer"
tempDir="/tmp/war-temp/"
deployDir="/tmp/war-deploy/"
watchDir="/tmp/war-listen/"
watchEnabled="false"/>
```

```
<ClusterListener
  className="org.apache.catalina.cluster.session.ClusterSessionListener"/>
</Cluster>
```

6. Edit the `/tomcat/conf/context.xml` and add a **distributable** parameter to the main **Context** element:

```
<Context distributable="true">
```

Copying WAR Files to the Nodes

Now that you have configured your application for clustering, you have to copy the Pentaho WARs to each node. Follow the instructions below.

1. If you have not already done so, install an identical version of Tomcat or JBoss (including the configuration files) to each node in the cluster. Alternatively you can copy the Tomcat or JBoss directory you've been working with to each node and configure the operating system to start and stop these services when the computer boots and shuts down.
2. If they were not copied over in the previous step, ensure that the Pentaho WARs have been copied to the `/server/default/deploy/` or `/webapps/` directory: **pentaho.war**, **pentaho-style.war**.

You now have a populated cluster.


Starting and Testing the Cluster

Follow the below instructions to start the cluster and verify that it is working properly.

1. Start the solution database.
2. Ensure that the machine hosting the shared pentaho-solutions directory is available to each node.
3. Start the application server on each node.
4. Start Apache and your application server on the load balancer.
5. Test the cluster by accessing the BA Server through the load balancer's IP address, hostname, or domain name and commence whatever test procedure you have designed for this scenario.

Changing the Java VM Memory Limits

If you are running out of memory even though your server has a lot of RAM, you probably need to increase the resources allocated to the JRE instance that runs the Pentaho software you're trying to improve performance on. Adjusting the memory limit is an easy configuration change to make, but the instructions differ depending on the client tool or Web application server you're using. Refer only to the sections below that apply to your situation.

 **Note:** If you are running multiple Pentaho programs concurrently, the sum of their JVM maximum memory limits must not exceed the available RAM minus system overhead.

Increasing Memory Limits on Microsoft Windows with a Graphical Installation

By default, Tomcat has a relatively low memory allotment. This can cause out-of-memory errors in the BA Server from time to time. The below instructions will explain how to increase the memory so you don't get this error. Instructions are also included for renaming the tomcat6 executable file so that BA Server starts automatically.

These instructions are for those who installed Pentaho Business Analytics graphically (as opposed to manually) and are Windows users.

1. Go to the **Windows Search Box** and enter `C:\Program Files\pentaho\server\biserver-ee\tomcat\bin\shutdown.bat` to shutdown the Pentaho BA Server.
2. Type `services.msc` into the **Windows Search Box**.
3. Find the Pentaho Server name (**Pentaho BA Server** or **Pentaho BI Server**) and open it so you can find the **service name**. The **service name** should appear at the top of the first tab (**General**). It will be **pentahobaserver**.
4. Go into the bin file (`C:\Program Files\pentaho\server\biserver-ee\tomcat\bin\`) and rename the **tomcat6w.exe** file to match the **service name** (**pentahobaserver** or **pentahobiserver**). This will ensure that the server starts with the software.
5. After you have renamed the file, open it by double-clicking on it. This will not open the file, it will allow you to edit it. The **Properties Window** will open.
6. Select the **Java** tab.
7. Set the memory setting to a minimum of **4096 M** and a maximum of **6144 M**, depending on your computer's memory capabilities.
8. Start the Tomcat server or service.

Your Tomcat server now has increased minimum and maximum memory limits. You can adjust the **JvmMx** number, which is a parameter that specifies the maximum limit, to a higher number if you prefer. However, if the Java virtual machine refuses to start with increased limits, then you will have to add more RAM to your system, stop some memory-intensive services, or reduce the maximum memory limit to a lower number. This problem occurs when there is not enough contiguous memory available to assign to the JVM.

Increasing Memory Limits on Linux with a Graphical Install

By default, Tomcat has a relatively low memory allotment. This can cause out-of-memory errors in the BA Server from time to time. To avoid errors, increase the memory allocation so you can start the BA Server automatically.

These instructions apply if you installed Pentaho Business Analytics using the graphical installer as opposed to manual installer and are Linux users.

1. Go to `/pentaho/server/biserver-ee/tomcat/bin/` directory and run the `./shutdown.sh` command to stop the appropriate server.
2. Change the directory to **biserver-ee/tomcat/scripts**.
3. Edit the **ctl.sh** file.
4. Locate the line under **start tomcat**, which looks like this: `export JAVA_OPTS="-Dpentaho.installed.licenses.file=/opt/pentaho/.insalledLicenses.xml -Xms128m Xmx768m -XX-MaxPermSize=256m -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000"`
5. Set the memory to a minimum of **4096 M** and a maximum of **6144 M**, depending on your computer's memory capabilities.

6. Start the Tomcat server or service.

Your Tomcat server now has increased minimum and maximum memory limits. You can adjust the `JvmMx` parameter to specify a higher maximum limit if you prefer. However, if the Java virtual machine refuses to start with increased limits, then you will have to add more RAM to your system, stop some memory-intensive services, or reduce the maximum memory limit to a lower number. This problem occurs when there is not enough contiguous memory available to assign to the JVM, and appears to happen more often on Microsoft Windows at lower thresholds than on other operating systems.

Increasing Memory Limits with an Archive or Manual Deployment

By default, Tomcat has a relatively low memory allotment. This can cause out-of-memory errors in the BA Server from time to time. To increase the memory limit, follow the below process.

1. Stop the Tomcat server or service.

2. **Because you are modifying your own Tomcat instance** and have performed a manual deployment of the BA Server WAR, edit the `~/ .bashrc` for the user account that starts the Tomcat service, or whatever configuration file or dialogue that contains global system variables on your BA Server machine. Set or modify the `CATALINA_OPTS` system variable to include reasonable minimum and maximum memory settings using the `-Xms` and `-Xmx` options.

```
export CATALINA_OPTS="-Xms4096m -Xmx6144m"
```

3. **If you are using a Pentaho-supplied Tomcat instance** provided in BA Server archive packages, edit the `start-pentaho` scripts (`.bat` for Windows, and `.sh` for Linux), and modify the `CATALINA_OPTS` environment variable, adjusting the values of `Xms` and `Xmx` in the same manner as the previous step.

```
export CATALINA_OPTS="-XMs4096m -Xmx6144m -XX:MaxPermSize=6144m -
Dsun.rmi.dgc.client.gcInterval=3600000
-Dsun.rmi.dgc.server.gcInterval=3600000"
```

4. If you are modifying a Windows service for Tomcat, you must use the `tomcat6.exe` command to reconfigure the service parameters within a command line window. You can access Windows Services by going to the **Windows Start Menu** and typing `services` in the **Search Programs and Files** box. See the below example.

Windows (this only applies to a Windows service for Tomcat)

```
tomcat6 //US//Tomcat6 --JvmMs=4096m --JvmMx=6144m
```

5. Start the Tomcat server or service.

Your Tomcat server now has increased minimum and maximum memory limits. You can adjust the `JvmMx` number (this parameter specifies the maximum limit) to a higher number if you prefer. However, if the Java virtual machine refuses to start with increased limits, then you will have to add more RAM to your system, stop some memory-intensive services, or reduce the maximum memory limit to a lower number. This problem occurs when there is not enough contiguous memory available to assign to the JVM, and appears to happen on Windows at lower thresholds than on other operating systems.

Increasing the Memory Limit in Aggregation Designer

Pentaho Aggregation Designer's startup script uses the default memory settings for your Java environment, which may be insufficient for your work. If you're experiencing an `OutOfMemory` exception, you must increase the amount of heap space available to Aggregation Designer by changing the Java options that set memory allocation. Follow the directions below to accomplish this.



Note: In the examples below, the memory size notations are **m** for megabytes and **g** for gigabytes. You can use whichever is most appropriate for your situation.

1. Exit Aggregation Designer if it is currently running.
2. Edit the `startaggregationdesigner` script and modify your Java command to include an `-Xmx` line that specifies a large upper memory limit.

Linux/Solaris shell script:

```
"$_PENTAHO_JAVA" $LICENSEPARAMETER -Xmx2g -jar "$DIR/lib/launcher-1.0.0.jar"
```

Windows batch file:


```
"%_PENTAHO_JAVA%" %LICENSEPARAMETER% -Xmx2g -jar "%~dp0lib\launcher-1.0.0.jar"
```

3. Start Aggregation Designer and ensure that there are no memory-related exceptions.

The Java virtual machine instance that Aggregation Designer uses now has access to more heap space, which should solve OutOfMemory exceptions and increase performance.

Increasing the Memory Limit in PDI

Pentaho Data Integration's startup script uses the default memory settings for your Java environment, which may be insufficient for your work. If you're experiencing an OutOfMemory exception, you must increase the amount of heap space available to PDI by changing the Java options that set memory allocation. Follow the directions below to accomplish this.

 **Note:** In the examples below, the memory size notations are **m** for megabytes and **g** for gigabytes. You can use whichever is most appropriate for your situation.

1. Exit Spoon if it is currently running.
2. Edit your Spoon startup script and modify the **-Xmx** value in the **OPT** variable so that it specifies a large upper memory limit.

Linux/Solaris shell script:

```
OPT="$OPT -Xmx2g -Xms256m -XX:MaxPermSize=128m -Djava.library.path=$LIBPATH -
DKETTLE_HOME=$KETTLE_HOME -DKETTLE_REPOSITORY=$KETTLE_REPOSITORY -DKETTLE_USER=
$KETTLE_USER -DKETTLE_PASSWORD=$KETTLE_PASSWORD -DKETTLE_PLUGIN_PACKAGES=
$KETTLE_PLUGIN_PACKAGES -DKETTLE_LOG_SIZE_LIMIT=$KETTLE_LOG_SIZE_LIMIT"
```

Windows batch file:


```
set OPT="-Xmx2g" "-XX:MaxPermSize=256m" "-Djava.library.path=%LIBSPATH%"
"-DKETTLE_HOME=%KETTLE_HOME%" "-DKETTLE_REPOSITORY=%KETTLE_REPOSITORY%"
"-DKETTLE_USER=%KETTLE_USER%" "-DKETTLE_PASSWORD=%KETTLE_PASSWORD%" "-
DKETTLE_PLUGIN_PACKAGES=%KETTLE_PLUGIN_PACKAGES%" "-DKETTLE_LOG_SIZE_LIMIT=
%KETTLE_LOG_SIZE_LIMIT%"
```

3. Start Spoon and ensure that there are no memory-related exceptions.

The Java virtual machine instance that PDI uses now has access to more heap space, which should solve OutOfMemory exceptions and increase performance.

Increasing the Memory Limit in Report Designer

Pentaho Report Designer's startup script uses the default memory settings for your Java environment, which may be insufficient for your work. If you're experiencing an OutOfMemory exception, you must increase the amount of heap space available to Report Designer by changing the Java options that set memory allocation. Follow the directions below to accomplish this.

 **Note:** In the examples below, the memory size notations are **m** for megabytes and **g** for gigabytes. You can use whichever is most appropriate for your situation.

1. Exit Report Designer if it is currently running.
2. Edit the **report-designer** script and modify the value of **-Xmx** to allocate more memory to Report Designer's JVM instance.

Linux/Solaris shell script:

```
"$_PENTAHO_JAVA" "-Dpentaho.installed.licenses.file=$PENTAHO_INSTALLED_LICENSE_PATH"
-XX:MaxPermSize=256m -Xmx2g -jar "$DIR/launcher.jar" $@"
```

Windows batch file:

```
set OPT="-XX:MaxPermSize=256m" "-Xmx2g"
```

3. Start Report Designer and ensure that there are no memory-related exceptions.

The Java virtual machine instance that Report Designer uses now has access to more heap space, which should solve OutOfMemory exceptions and increase performance.

Increasing the Memory Limit in Weka

Weka uses the memory settings passed to it from the Java command line or the script that invokes it. If you're experiencing an OutOfMemory exception, you must increase the amount of heap space available to Weka by changing the Java options that set memory allocation. Follow the directions below to accomplish this.



Note: In the examples below, the memory size notations are **m** for megabytes and **g** for gigabytes. You can use whichever is most appropriate for your situation.

1. Exit Weka if it is currently running.
2. If you are running Weka standalone from the command line, modify your Java command to include an **-Xmx** line that specifies a large upper memory limit.

```
java -Xmx2g weka.jar
```

3. If you are running Weka as part of a script, change your Java invocation so that it includes the above **-Xmx** setting.
4. Start Weka and ensure that there are no memory-related exceptions.

The Java virtual machine instance that Weka uses now has access to more heap space, which should solve OutOfMemory exceptions and increase performance.