HITACHI
Inspire the Next

# Monitoring Pentaho Analyzer Performance

Change log (if you want to use it):

| Date | Version | Author | Changes |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

# Contents

# Overview

This document covers some guidelines on monitoring Pentaho Analyzer performance. You will learn about monitoring Analyzer Performance and identifying the root cause of performance issues.

Our intended audience is Pentaho administrators, or anyone with a background in database use or administration who is interested in dynamically exploring data and monitoring to increase performance speed.

| Software | Version(s) |
|---|---|
| Pentaho | 6.x, 7.x, 8.0 |
| Mondrian | 3.x |

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

## Before You Begin

This document assumes that you have knowledge about Pentaho Analyzer and Mondrian and that you have already installed and configured Pentaho server and client tools. More information about related topics outside of this document can be found at Pentaho Analyzer External JavaScript API Reference and Customize Pentaho Analyzer.

### *Use Case: Slow Performance of Analyzer Report*

*Janice is an analyst working with the Pentaho software. Users have complained to her that working with Analyzer is very slow, and Janice needs to track down the reasons why that is.*

*Janice will look at the log files and `pro_audit` table to explore the performance of requests to her data source, so that she can see what is slowing things down.*

# Introduction to Analyzer Performance Tuning

Pentaho Analyzer, based on the Mondrian engine, provides you with analysis of historical trends and data by letting you view key performance metrics.

Mondrian responds to queries fast enough to allow users to interactively explore business data by drilling and cross-tabulating information.

## Maximizing Performance

When you make a change to a report such as adding or removing a dimension or measure, you need the report to update quickly. Mondrian optimizes performance to make sure you get your results, and it can be further customized for even better results.

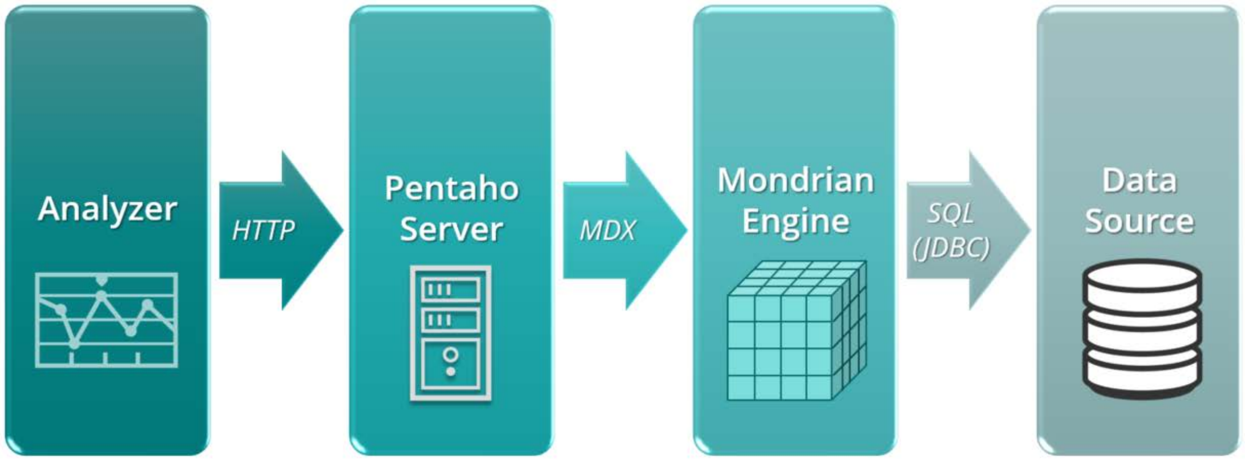

*Figure 1: Analyzer Architecture*

*Table 1: Analyzer Architecture Description*

| Component | Description |
|---|---|
| **Analyzer** | Each time you drag and drop a level or measure into Analyzer, it sends a request to the server. |
| **Pentaho Server** | The server will generate an MDX query to the Mondrian engine. |
| **Mondrian Engine** | The Mondrian engine receives the query and checks its internal cache for any results already held in memory in the segment cache. If the data is not found in the segment cache, the Mondrian engine will check the aggregate table for it. If the data is not found in the aggregate table, the Mondrian engine will then check the data source. |
| **Data Source** | Data returned from the data source is returned to the Mondrian engine and cached internally to fulfill future data requests. |

# Analyzer Performance Tuning

*Janice needs to investigate performance issues that could be in any one (or more) of the four tiers of Analyzer, Pentaho Server, the Mondrian engine, or the data source. She will need to set up monitoring to figure out the cause(s).*

Once you have set up the plugin, you will want to monitor its performance. When you discover problems related to Analyzer during monitoring, you will then need to tune whichever tier you found to be the root cause. You can find more information on these topics in the following sections:

- Monitor Performance
- Gather Execution Statistics
- Identify Root Cause

## Monitor Performance

Analyzer will log the start and the end of each request to the log file. In addition, the Mondrian engine can log both the MDX statements sent to the Mondrian engine and the SQL generated and sent to the data source. Like many Java applications, Pentaho Server uses the common logging framework log4j. Although this type of logging is turned off by default, it can be enabled by configuring the log4j files. However, whether logging is turned on or not, what is in the existing log4j file should not be used.

Instead, you need to enable MDX, SQL, and the `RolapUtil` in one file. You will not be able to do the analysis you need to do if this is not set up properly.

1. Stop the Pentaho Server.
2. Open `log4j.xml` from its location; for example, `C:\Pentaho8.0\server\pentaho-server\tomcat\webapps\pentaho\WEB-INF\classes\`
3. Comment out any previous sections of the file related to Mondrian.
4. Insert this code near the end of the file and note that the path in `param name="File"` needs to be the correct file path for your Pentaho installation. Then, save the file and restart the Pentaho Server.

```
<!-- Mondrian -->
   <appender name="MONDRIAN"
class="org.apache.log4j.DailyRollingFileAppender">
     <param name="File" value="/home/pentaho/pentaho/server/biserver-
ee/logs/mondrian.log"/>
     <param name="Append" value="true"/>

     <param name="DatePattern" value="'.'yyyy-MM-dd"/>
     <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%d %-5p (%t) [%c] %m%n"/>
     </layout>
   </appender>

   <category name="mondrian.mdx" additivity="false">
      <priority value="DEBUG"/>
      <appender-ref ref="MONDRIAN"/>
```

```
</category>

<category name="mondrian.sql" additivity="false">
   <priority value="DEBUG"/>
   <appender-ref ref="MONDRIAN"/>
</category>

<category name="mondrian.rolap.RolapUtil" additivity="false">
   <priority value="DEBUG"/>
   <appender-ref ref="MONDRIAN"/>
</category>
```

To collect your execution information, you will need to confirm that auditing in the `pro_audit` table is enabled:

1. Open the `/pentaho-solutions/system/pentahoObjects-spring.xml` file with a text editor.
2. Locate the `IAuditEntry` bean and make sure it reads this way:

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

3. If the bean says `core.audit.NullAuditEntry` instead of `services.audit.AuditSQLEntry`, change the bean to the line above.
4. Save and close the file.

Now, run your normal reports and executions for a while so that you can get enough sample data. Try several different iterations of the reports you are experiencing problems with.

## Gather Execution Statistics

Next, to gather your execution statistics, you will need to run a query or report on top of the Hibernate `pro_audit` table to see which reports and xanalyzer files are running the slowest.

The query used in the example here is a simple version that will give the basic details on any database type. It will produce a list of all Analyzer reports and their executions statistics sorted by the longest running, in descending order, and who ran which Analyzer report when, and how long it took to execute.

1. Run this query on top of the Hibernate `pro_audit` table:

```
select obj_id, actor, message_name, duration, audit_time
from public.pro_audit
where job_id = 'Analyzer' and message_type = 'instance_end'
order by duration desc
```

2. Use this information to retrieve additional details in the Mondrian and Pentaho log files to pinpoint the root cause of why that Analyzer report was slow.

**Optional:** Use a Detail Query to gather additional details such as the date components, the schema and cube that the query was run against, the levels and measures used in the query. This way, not only can you determine what queries are slow, but also which cubes and schemas have slower queries.

## Identify Root Cause

Once you identify the slow queries, you can get details in the Mondrian log files to identify what step in the process took the longest time:

1. For each report that is running slow or beyond expectations, determine the request ID. In the Mondrian log file, the column `MESSAGE_NAME` contains the request ID for Mondrian.
2. Get rows with the request ID matching the `pro_audit.message_name`.
3. For each Mondrian request, there are at least two entries with a request ID in sequence, each with times. One is the start of the request, and one is the ending summary.
    a. The first entry in the `Mondrian.log` file has the MDX associated with the request. This is helpful to see what was requested.
    b. For the second entry, each line has the start time and the request ID it is related to.
4. Each line also has duration timings in milliseconds.
    a. Examine each portion of the total to see what should be investigated.
    b. Start with the portion with the highest percentage of time, which is usually the execute or parsing.

*Table 2: Mondrian Log Line Items*

| Line Item | Description |
|---|---|
| `Started` | Point in time when Analyzer started processing this report request. |
| `SetupAxis` | Duration it took for Analyzer to create each axis for a report (possibly executing MDX to generate calculated members). |
| `GeneratedMDX` | Duration it took for Analyzer to generate MDX for each axis (possibly executing MDX to evaluate filters). |
| `ParsedMondrian` | Duration it took Mondrian to parse the MDX statements. |
| `ExecutedMondrian` | Duration it took for Mondrian to execute the MDX statement. *This number is the culprit 80-95% of the time.* When this number is high, you will need to tune the database, add aggregate tables, or enable proactive caching. The actual SQL statements sent to the database will be in the Mondrian log between the start of this request and the end of this request, so you can use this method to find the SQL statement that is the culprit. |
| `Sorted` | Duration it took for Analyzer to complete sorting each axis. |
| `Rendered` | Duration it took for Analyzer to render the report to the desired format type. |
| `CacheAccess` | Duration it took Analyzer to retrieve the `AnalyzerResultSet` from the cache. |
| `Total` | Total duration from start to finish. |

Now that you have the results, you can take the appropriate steps to resolve the performance issues. You can talk to Customer Support, check our documentation on Analyzer, find information on how to Manage the Pentaho Server, try some Mondrian Log Analysis, or look into general or database performance tuning.

## Related Information

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

- Manage the Pentaho Server
- Mondrian Log Analysis
- mondrian_log4j.xml
- Pentaho Analyzer
- Pentaho Components Reference
- Pentaho Customer Support
- Performance Tuning

## Finalization Checklist

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

Name of the Project:____________________________________________________________

Date of the Review:_____________________________________________________________

Name of the Reviewer:___________________________________________________________

| Item | Response | Comments |
|---|---|---|
| Did you set up a test environment and prepare your data? | YES_______ NO_______ | |
| Did you run queries in your test environment and capture your performance metrics? | YES_______ NO_______ | |
| Did you tune your database? | YES_______ NO_______ | |
| Did you tune the Mondrian schema? | YES_______ NO_______ | |

# Appendix: Detail Query

Use this text for a Detail Query to find more information about which cubes and schemas have slower queries:

```
SELECT Inst_Srt.AUDIT_ID AS START_ID
     ,concat_ws('_',Inst_Srt.INST_ID,Inst_Srt.MESSAGE_Name) as Inst_Key
     , Inst_Srt.AUDIT_TIME AS START_TIME
     , Inst_End.AUDIT_TIME AS END_TIME
     , DATE(Inst_Srt.AUDIT_TIME) AS DAY
     , extract (YEAR from Inst_Srt.AUDIT_TIME) AS YEAR
     , extract (QUARTER from Inst_Srt.AUDIT_TIME) AS QUARTER
     , extract (MONTH from Inst_Srt.AUDIT_TIME) AS MONTH
     , CASE extract (month from Inst_Srt.AUDIT_TIME)
      WHEN 1 THEN 'Jan'
      WHEN 2 THEN 'Feb'
      WHEN 3 THEN 'Mar'
      WHEN 4 THEN 'Apr'
      WHEN 5 THEN 'May'
      WHEN 6 THEN 'Jun'
      WHEN 7 THEN 'Jul'
      WHEN 8 THEN 'Aug'
      WHEN 9 THEN 'Sep'
      WHEN 10 THEN 'Oct'
      WHEN 11 THEN 'Nov'
      ELSE 'Aug' END AS MONTH_NAME
     , extract (week from Inst_Srt.AUDIT_TIME) AS WEEK
     , extract (day from Inst_Srt.AUDIT_TIME) AS DAY_OF_MONTH
     , extract (dow from Inst_Srt.AUDIT_TIME) AS DAY_OF_WEEK
     , CASE extract (dow from Inst_Srt.AUDIT_TIME)
      WHEN 0 THEN 'Sun'
      WHEN 1 THEN 'Mon'
      WHEN 2 THEN 'Tue'
      WHEN 3 THEN 'Wed'
      WHEN 4 THEN 'Thu'
      WHEN 5 THEN 'Fri'
      ELSE 'Sat' END AS DAY_NAME
     , extract (HOUR from Inst_Srt.AUDIT_TIME) AS HOUR
     , extract (MINUTE from Inst_Srt.AUDIT_TIME) AS MINUTE
     , CASE WHEN Inst_End.MESSAGE_TEXT_VALUE='HTML' THEN
Inst_End.DURATION/1000 ELSE Inst_End.DURATION END AS DURATION
     , Inst_Srt.OBJ_ID AS FULL_FILE_NAME
     , coalesce(Inst_Srt.ACTOR,'UNKNOWN') AS USER
     --, coalesce(Inst_Srt.MESSAGE_TEXT_VALUE, 'NONE') AS ANALYZER_QUERY
     , (string_to_array(Inst_Srt.MESSAGE_TEXT_VALUE, '~', 'None'))[1] as
analyzer_schema
     , (string_to_array(Inst_Srt.MESSAGE_TEXT_VALUE, '~', 'None'))[2] as
analyzer_cube
     , array_to_string(ARRAY(SELECT
UNNEST(string_to_array(Inst_Srt.MESSAGE_TEXT_VALUE, '~', 'None')) LIMIT ALL
OFFSET 2), '~') as analyzer_levels
     , CASE WHEN Inst_Srt.JOB_ID='com.pentaho.iadhoc.service.f' THEN 'I
Adhoc'
            WHEN right(Inst_Srt.OBJ_ID,5) = '.xcdf' THEN 'CDF Dashboard'
            ELSE substring(Inst_Srt.OBJ_ID from 0 for
length(Inst_Srt.OBJ_ID)- position('.' in reverse(Inst_Srt.obj_id))+1) END
AS ACTION_TYPE
     , CASE WHEN Inst_Srt.OBJ_ID ='New Analyzer Report' THEN
Inst_Srt.OBJ_ID
            ELSE right(Inst_Srt.OBJ_ID,position('/' in
reverse(Inst_Srt.obj_id))-1) END AS FILE_NAME
     , CASE WHEN Inst_Srt.OBJ_ID ='New Analyzer Report' THEN
Inst_Srt.OBJ_ID ELSE substring(Inst_Srt.OBJ_ID from
(length(Inst_Srt.OBJ_ID)- position('.' in reverse(Inst_Srt.obj_id))+1)) END
AS FILE_TYPE
     , CASE WHEN Inst_Srt.OBJ_ID ='New Analyzer Report' THEN
Inst_Srt.OBJ_ID ELSE left(Inst_Srt.OBJ_ID,length(Inst_Srt.OBJ_ID)-
position('/' in reverse(Inst_Srt.obj_id))) END AS FILE_PATH

FROM
  public.pro_audit Inst_Srt
LEFT OUTER JOIN public.pro_audit Inst_End
ON Inst_Srt.INST_ID = Inst_End.INST_ID AND Inst_Srt.MESSAGE_NAME =
Inst_End.MESSAGE_NAME
WHERE
  Inst_Srt.MESSAGE_TYPE = 'instance_start'
  AND Inst_End.MESSAGE_TYPE = 'instance_end'
  and Inst_Srt.job_id = 'Analyzer'
ORDER BY duration desc
```