Denodo Load Testing with Apache JMeter

Revision 20210830

Denodo North America & APAC: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA
Denodo Iberia & Latino América: Montalbán 5, 28014 Madrid, Spain
Denodo EMEA: 21st Floor, Portland House, Bressenden Place, London SW1E 5RS. UK
Denodo DACH: Karlstraße 10, 80333 München. Germany
www.denodo.com

CONTENTS

1 INTRODUCTION

This document explains how to configure Apache JMeter (http://jmeter.apache.org/index.html) to carry out performance tests on the Denodo Platform.

Apache JMeter is an open source performance testing tool. Any application that works on request/response model can be load tested with JMeter. It offers modules to test databases using JDBC, web services, web pages and many more. It can be used to simulate a heavy load on a server to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server under heavy concurrent load.

Therefore, it is a perfect tool to test a Denodo installation simulating access using JDBC or published web services, and tune the configuration parameters of the server accordingly. It is also a very useful tool to perform capacity analysis on a production server and draw the red lines for the current deployment.

2 INSTALLING JMETER

JMeter can be downloaded from http://jmeter.apache.org/download_jmeter.cgi.

No special installation is required, just decompress the file. JMeter is a Java based application, so you will need to have the java executable in your OS path.

2.1 LOADING THE JDBC DRIVER

In order to use a JDBC connection to Denodo, the Denodo JDBC driver has to be loaded in the JMeter classpath. To do so, just copy Denodo's JDBC Driver to `<JMeter>/lib` folder.
Denodo's JDBC driver can be found under `<DENODO_HOME>/tools/client-drivers/jdbc/denodo-vdp-jdbcdriver.jar` (For Denodo versions previous to 6.0 it will be located under `<DENODO_HOME>/lib/vdp-jdbcdriver-core/denodo-vdp-jdbcdriver.jar`)

2.2 RUNNING JMETER

To launch JMeter GUI, just execute the file `<JMeter>/bin/jmeter.bat or jmeter.sh`.

2.3 OTHER INFORMATION

This guide is not intended to be an exhaustive list of all the features of JMeter. For detailed information please refer to the Apache JMeter manual, that can be found online in http://jmeter.apache.org/usermanual/index.html.

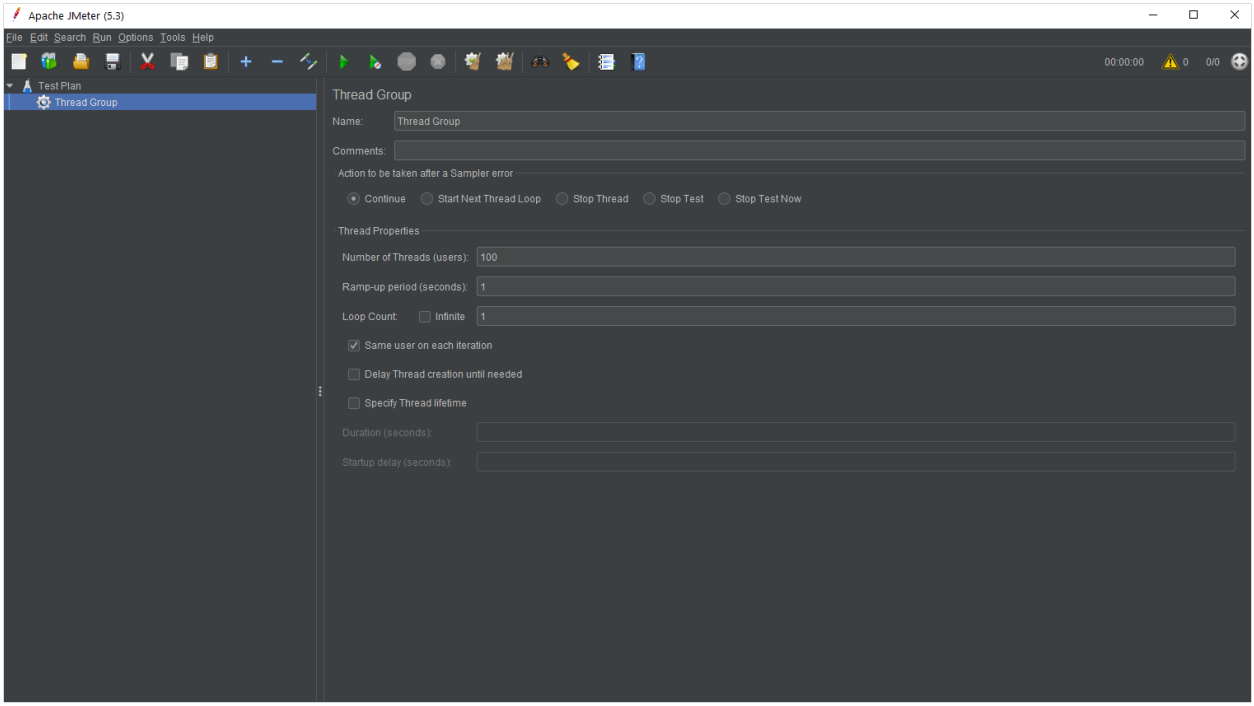
3 SETTING UP THE TESTING PLAN

In JMeter a test plan is defined as a number of concurrent users that execute the defined operations, in our case JDBC or web services requests. To construct the test plan, we will need the following elements:

- Thread group (the users)
- JDBC and/or HTTP Request
- Report Listener

3.1 ADDING USERS

The first step in a JMeter Test Plan is to add a Thread Group element. The Thread Group tells JMeter the number of users you want to simulate, how often the users should send requests, and the how many requests they should send. To add the Thread Group element, select the Test Plan, click your right mouse button to get the Add menu, and then select `Add > Thread(Users) > Thread Group`.



In the section Thread Properties, configure the following parameters:

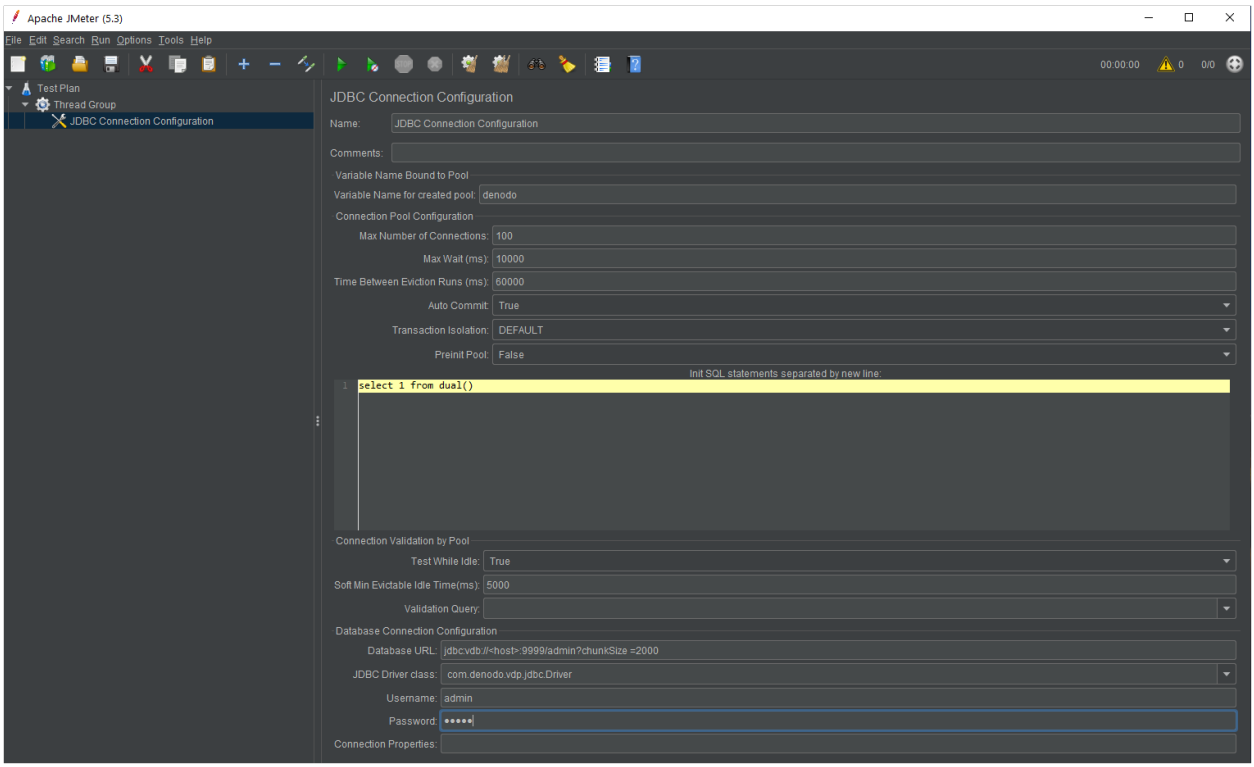
- **Number of Threads (users)**: This property represents the number of concurrent users that will be tested. Start with a value close to the amount of concurrent users using (or planning to use) the system at the moment, and then increase this value to try to determine the limits of the current set up.
- **Ramp up Period (in seconds)**: This property tells JMeter how long to delay between starting each user. For example, if you enter a Ramp-Up Period of 5 seconds, JMeter will finish starting all of your users by the end of the 5 seconds. So, if we have 5 users and a 5 second Ramp-Up Period, then the delay between starting users would be 1 second (5 users / 5 seconds = 1 user per second). If you set the value to 0, then JMeter will immediately start all of your users. It is usually a good idea to add some seconds here so the different queries defined in the following sections are not executed always at the same time in blocks.
- **Loop Count**: This property tells JMeter how many times to repeat your test. To simulate a high load during a longer period of time add a value greater than one. This will provide us with a more realistic scenario. Usually 3 to 5 is a good number.

3.2 ADDING JDBC REQUESTS

Now that we have defined our users, it is time to define the tasks that they will be performing. In this case, JDBC requests to the Denodo server.

3.2.1 Setting up the JDBC connection

Before setting up the requests we will need to configure the JDBC Driver and the connection pool. Begin by selecting the Users element. Click your right mouse button to get the Add menu, and then select `Add > Config Element > JDBC Connection Configuration`. Then, select this new element to view its Control Panel.



Configure the fields as follows:

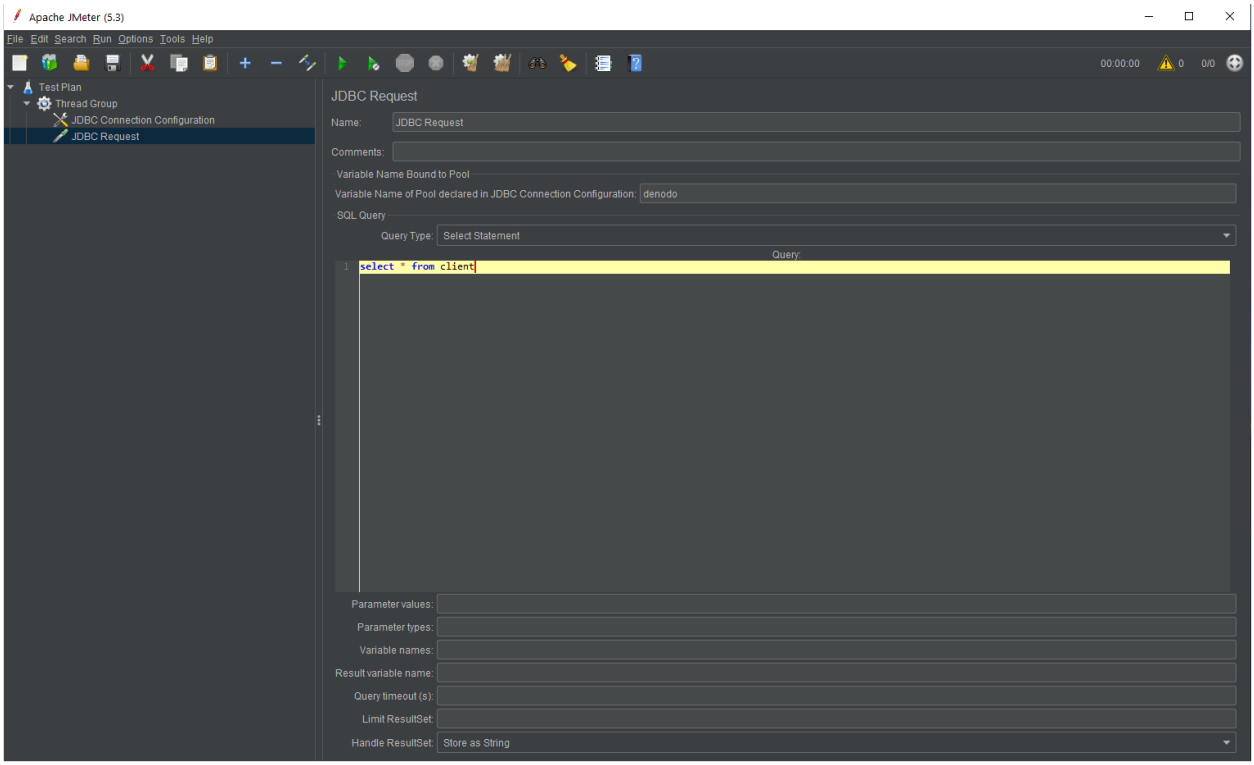
- **Variable name bound to pool**: This needs to uniquely identify the configuration. It is used by the JDBC Sampler to identify the configuration to be used.
- **Database URL**: jdbc:vdb://<host>:9999/<database>?chunkSize =<value>
  - ***chunkSize***: This parameter establishes the maximum number of results that a block can contain. When the Server obtains enough results to complete a block, it sends this block to the driver and continues processing the next results. The section Parameters of the JDBC Connection URL of the Virtual DataPort Administration Guide provides more information regarding this parameter and others for the JDBC connection).
- **JDBC Driver class**: com.denodo.vdp.jdbc.Driver
- **Username**: test user.
- **Password**: password for test user.
- **Validation query**: Select 1
- **Max Number of Connections**: This property represents the maximum size of the connection pool. Start with the value currently set up in your client application, and modify it if needed based on the results of your tests.
- The rest of the parameters can be left with their default values.

3.2.2 Defining the requests

Select the Users element again. Click your right mouse button to get the Add menu, and then select `Add > Sampler > JDBC Request`.

You can define several requests for your test plan. JMeter will send the requests in the order you add them to the tree, and will execute them for each user. It is usually a good

idea to choose a set of queries from a previous analysis based on the monitoring of the current production scenario.



You will have to define the following parameters:

- **Name**: use a clear name to identify the request in the test plan.
- **Variable Name Bound to Pool**: use the value defined in the JDBC Configuration screen to use the connection pool defined there.
- **SQL Query**: add the query you want to test and choose the Query Type (probably a Select Statement) in the drop down menu.

3.3 ADDING HTTP REQUESTS

If you want to test the access to Denodo via Web Services, you will need to set up an HTTP Request sampler or a Web Service (SOAP) Request, depending on your case. The configuration is very similar to the JDBC Requests, although you don't need to configure any driver in this case.

You can also mix JDBC and web service requests in the same test case to simulate a scenario with mixed load. Just keep in mind that the requests will be sent in the same order they are defined in the test plan tree.

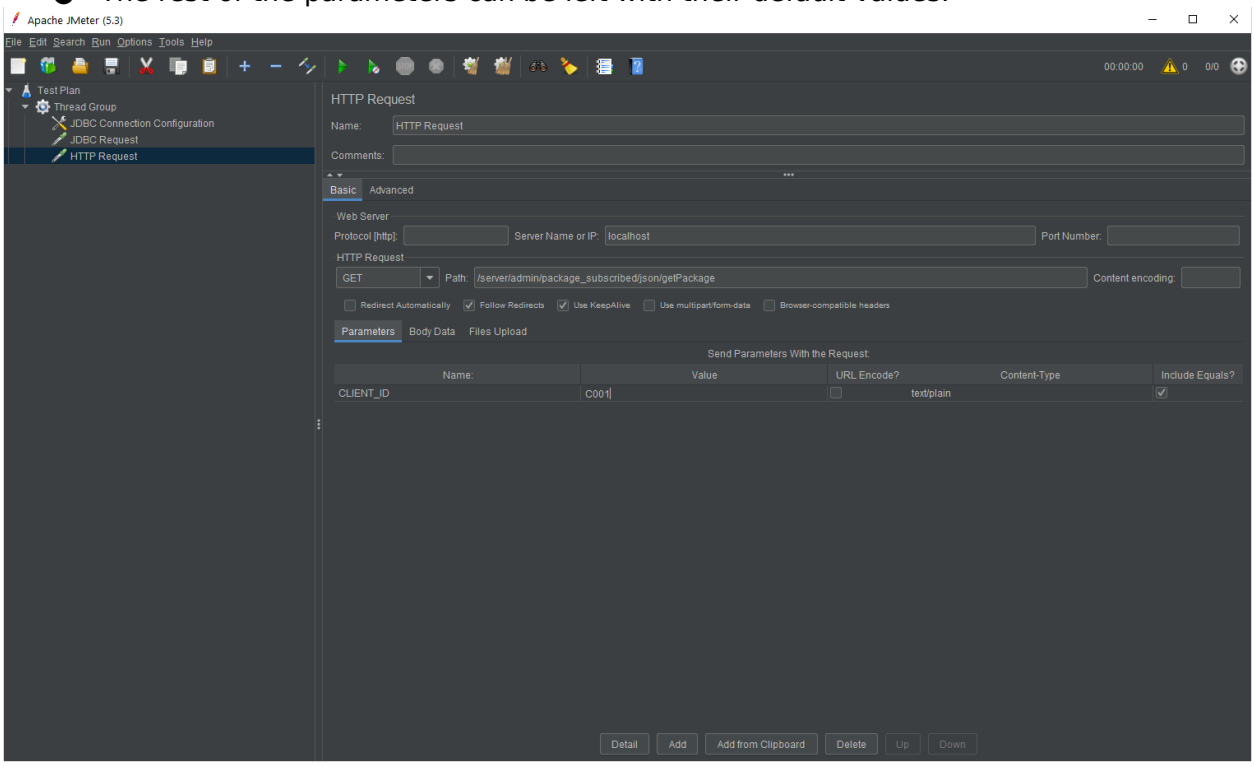
To add a HTTP request select the Users element again. Click your right mouse button to get the Add menu, and then select `Add > Sampler > HTTP Request`.

3.3.1 HTTP Requests to test REST (XML or JSON) access

- Name: change the name to something that identifies this particular request.
- Web Server

- Server Name or IP: server name where request will be sent.
- Port number: port where the web container will be listening. In Denodo, 9090 is the default port for the web container.

- Http Request
  - Path: path to the web service, for example: `/server/admin/package_subscribed/json/getPackage`.
  - Method: select the method from the drop down, usually GET.
  - Parameters: if the call requires input parameters, you can add them here.
- The rest of the parameters can be left with their default values.



3.4 ADD A LISTENER TO GET A REPORT FOR THE TEST RESULTS

The final element you need to add to your Test Plan is a Listener. This element is responsible for storing all of the results of your requests in a file and presenting a visual model of the data.

Select the Users element and add a Summary Report listener (`Add > Listener > Summary Report`).

4 RUNNING THE TEST

Important Note: The Apache JMeter documentation recommends avoid using the GUI for load testing. The graphical interface mode does have a limitation which slows down the CPU utilization while running the recorded script.

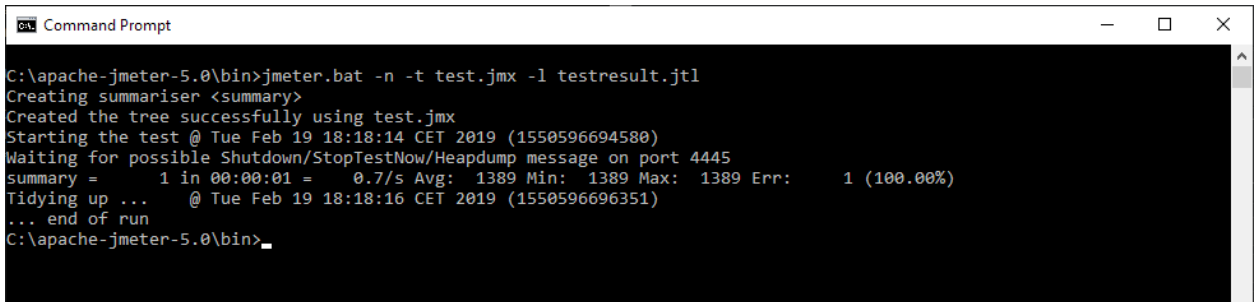
It is recommended to use the command line interface option instead of the GUI mode for running the load tests defined above.

4.1 RUNNING THE TEST USING THE COMMAND LINE INTERFACE

Once the plan is set up, you will have the tests stored in .jmx file. For instance: `test.jmx` file.

You can execute the plan using a command line and execute a command like the following:

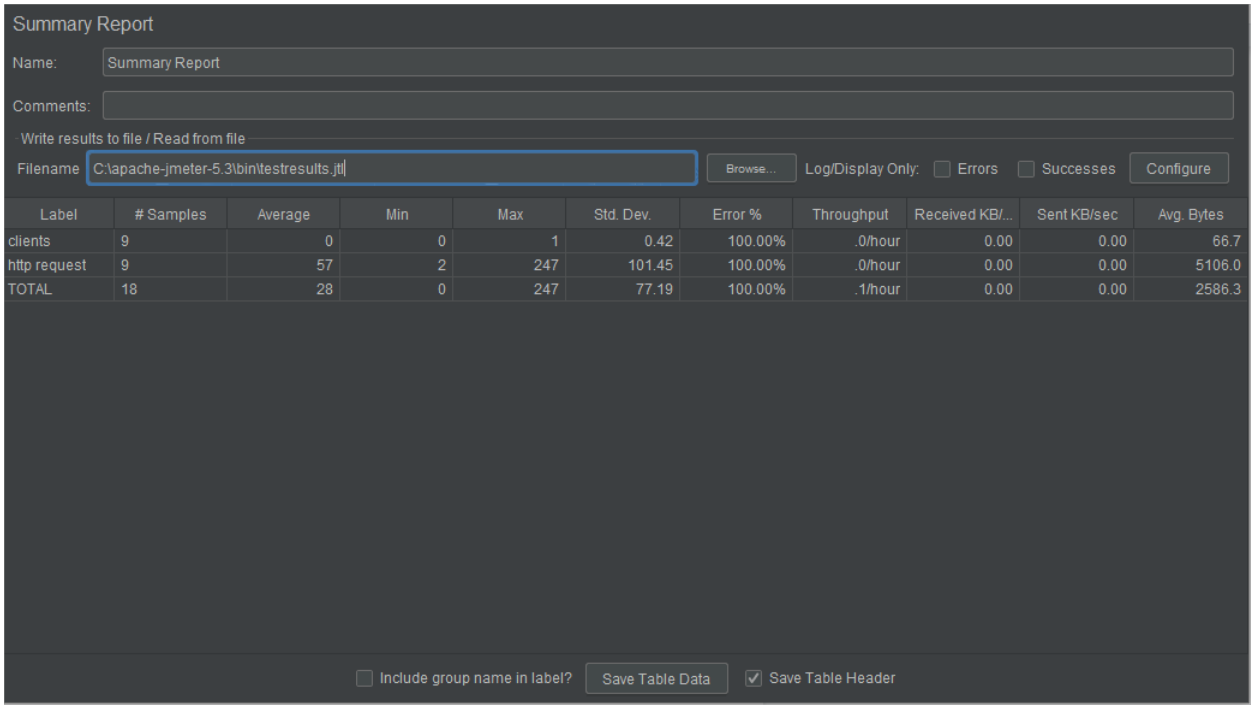
```
<JMeter>/bin/jmeter.bat -n -t test.jmx -l testresults.jtl
```



The command will generate a `testresults.jtl` file in the `<JMeter>/bin` folder.

After generating the `testresults.jtl` file you can view the results following these steps:

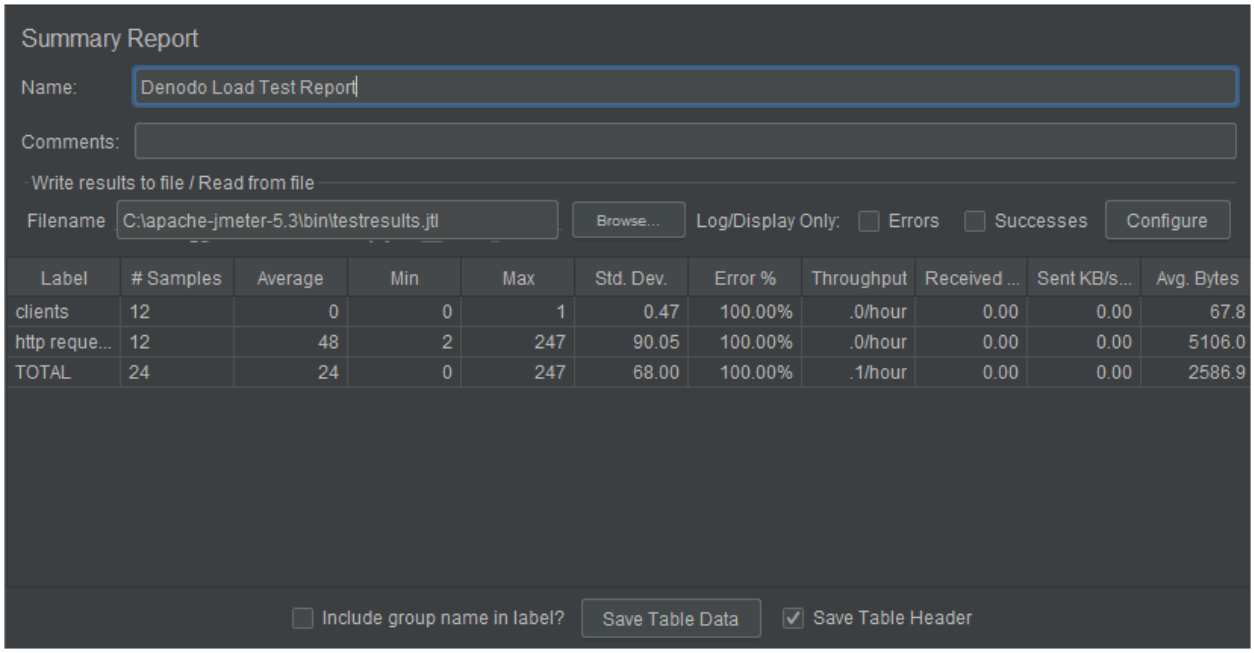
1. Open JMeter in GUI mode.
2. Add any Listener like the Summary Report.
3. Click `Browse…` in the `Filename` field to Read results from file.
4. Results from the file will appear in the listener.



4.2 RUNNING THE TEST USING THE GRAPHICAL INTERFACE

Once the test plan is set up, you can run it clicking the “Play” button in the toolbar, or using the menu option Run > Start.

Select the Summary Report element in the tree to see the report. In the report you will find useful metrics like the execution times (min, max and average), the percentage of errors and the throughput (queries per second).

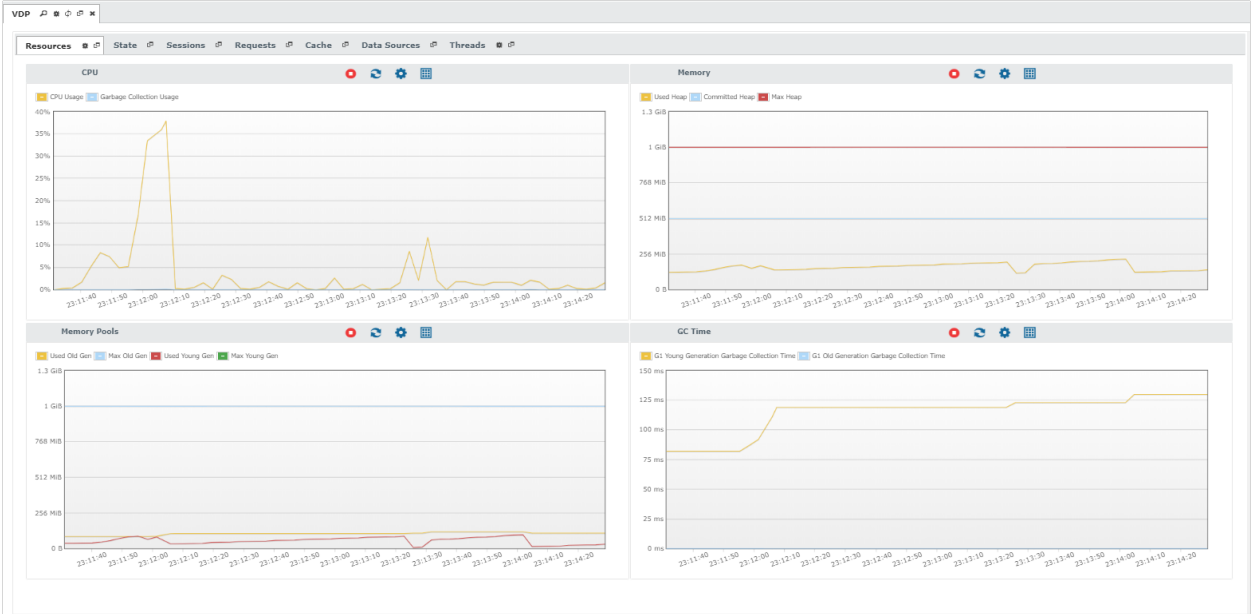


4.3 MONITORING THE BEHAVIOR OF THE DENODO SERVER

In combination with the report generated by JMeter, it is also very useful to monitor the behavior of the Denodo server in real time with tools like the Denodo Diagnostic & Monitoring tool or VisualVM.

This will allow you to monitor not only the results of the queries, but also a number of interesting parameters like:

- CPU usage.
- Memory (Heap space) usage.
- Memory Pools.
- GC Time.
- Status of the connection pools managed by Denodo to the data sources.



Denodo Diagnostic & Monitoring tool monitoring different tests launched from JMeter

5 ANALYZING THE RESULTS

Once the test is completed, it is time to analyze the results. If there are no errors in the execution, we can increase the number of users (in the configuration screen of the users element in JMeter) to test the server under a heavier load.

An analysis of several parameters in execution time (usually via JMX or using the Diagnostic & Monitoring tool) and the analysis of the logs of the server (found in `<DENODO_HOME>/logs/vdp/vdp.log`) will help in the tuning process.

5.1 TUNING DENODO SETTINGS

The following parameters are usually involved in the behavior of the system and may impact the result of the tests.

5.1.1 Java Virtual Machine memory settings

The Denodo server is a Java process, and as such, it needs a memory configuration adequate to its needs. In high load scenarios, the following parameters are recommended:

- `-server`: With this option, the JVM and its Garbage Collector (GC) are configured to run server applications.
- `-Xms4096m`: amount of allocated memory. Allocated memory is fully backed by physical memory. A heap size of 4GB is a good start point for a production deployment.
- `–Xmx4096m`: amount of reserved memory. The JVM announces to the OS that at some point, it may request this amount of memory, but it does not need it yet. The reserved memory can be used by other processes while is not allocated by

the JVM. Both parameters should have the same value in server-class applications.

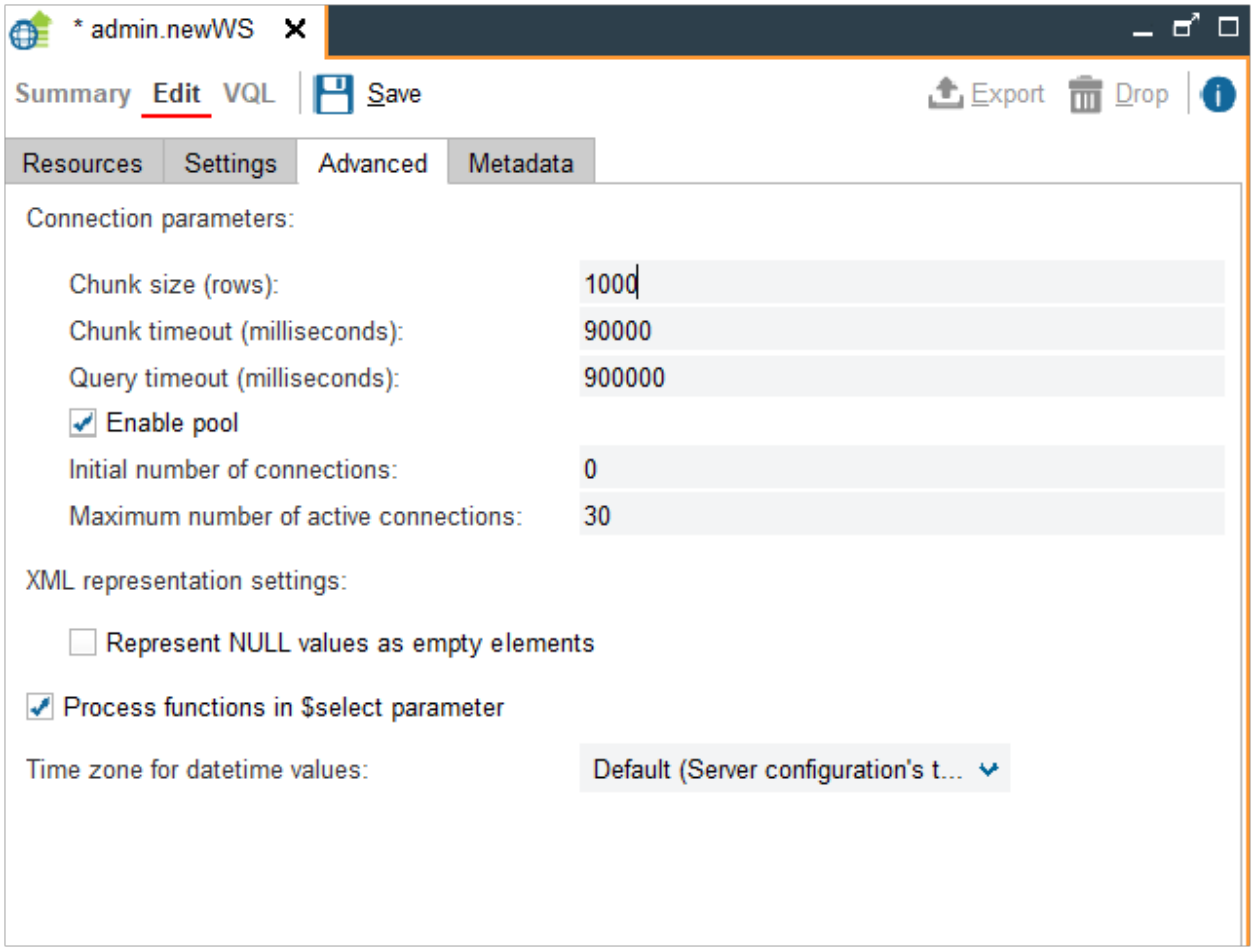
- `-XX:MaxPermSize=256m`: Top size of the Permanent Generation of the JVM's heap (for Denodo versions older than Denodo 7.0 only)
- `-XX:+DisableExplicitGC`: The JVM ignores all invocations to System.gc() from the code. The JVM still performs garbage collection when necessary.
- `-XX:+UseG1GC`: The JVM uses the G1 Garbage collector.
- `-XX:NewRatio`: NewRatio is the ratio between the “Young Generation” (eden + survivor spaces 1 and 2) and “Old Generation”. If NewRatio=4 and total heap size is 500, “Young Generation Size” will be 100 and “Old Generation Size” will be 400.
- `-XX:CMSInitiatingOccupancyFraction`: Informs the JVM when the CMS should be triggered. It allows to create a buffer in the heap that can be filled with data while the CMS is working. The value is expressed as a percent and it should be calculated based on the speed in which memory is consumed in the old generation space during the expected load.
- `-XX:ReservedCodeCacheSize`: This parameter defines the size of the region of the heap in which the JVM stores compiled native code

When the server runs out of memory, it will write Heap Space errors in the log file.

For for information about the recommended settings for the JVM, you can refer to the following Knowledge Base article: Denodo Admin and Development Best Practices

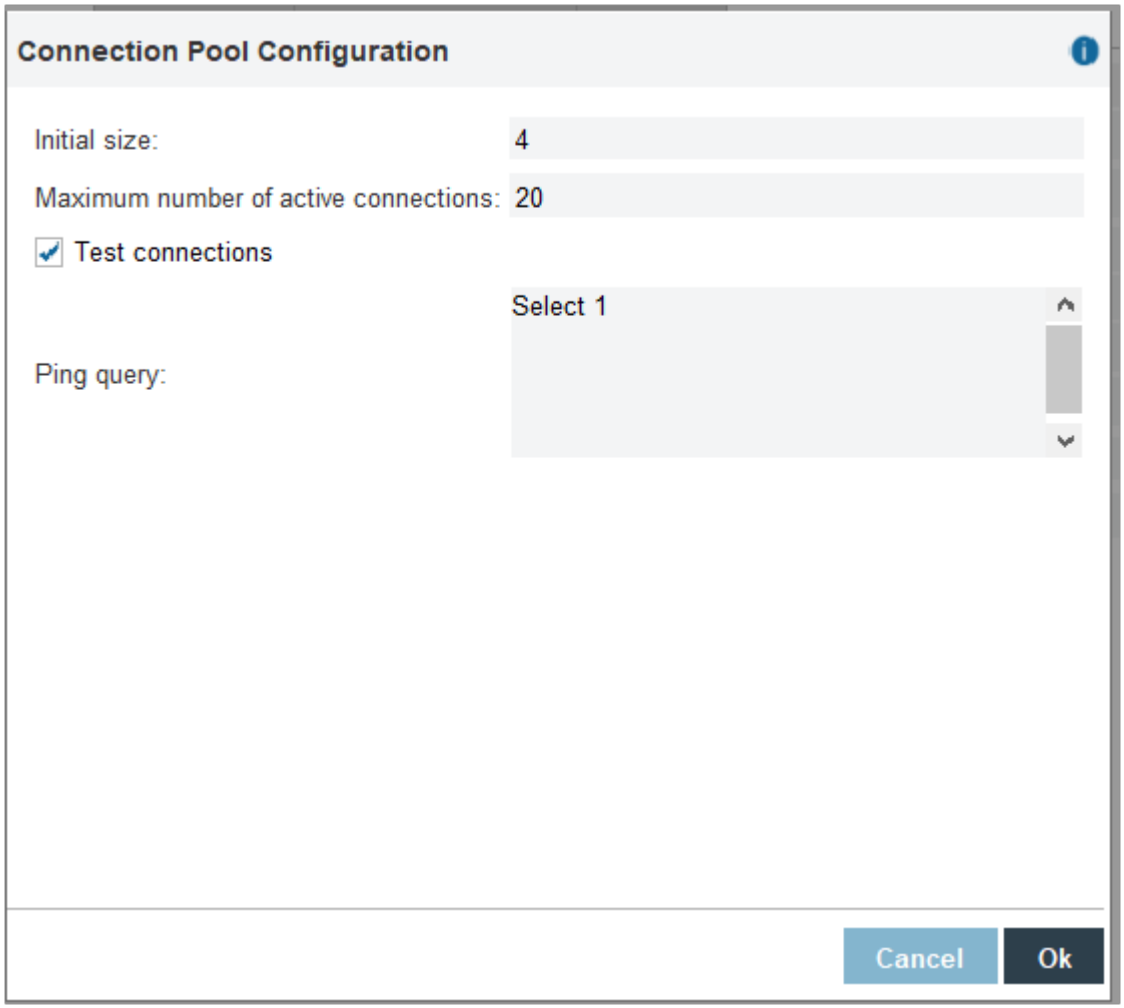
5.1.2 Client connections pool

If the client application is connected to Denodo using a connection pool, their parameters must be set accordingly. In the case of JMeter, this is configured in the JDBC configuration screen, as explained in previous sections. This also applies to the web services deployed in the embedded web service container that can be configured from the Edit Web service screen:



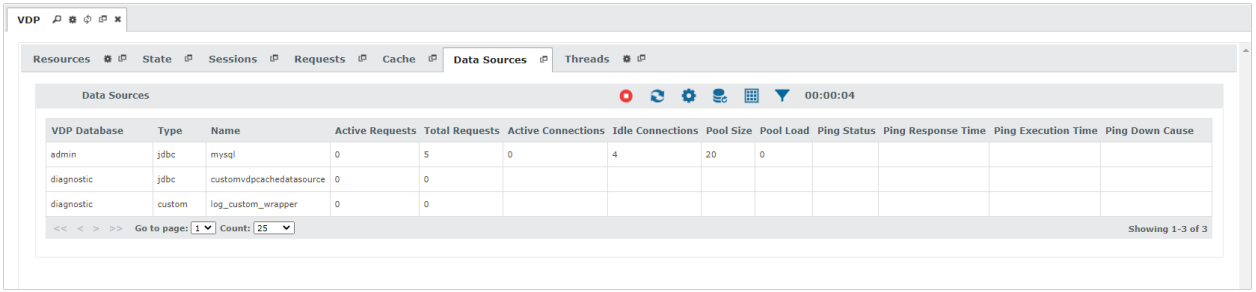
5.1.3 Sources connections pools

JDBC and ODBC sources use a connection pool for the queries issued to these types of sources. The values of these parameters can limit the number of outbound connections, leaving new incoming queries waiting for an available connection. These parameters can be modified in the "`Configuration`" screen for the data source, clicking on "`Connection>Connection pool configuration`".



The status of the connection pools during the execution of the test can be accessed via JMX, using the properties numActive and numIdle in the beans `com.denodo.vd.management.mbeans.DataSourceManagementInfo.<db>.(jdbc|odbc).<data source>`

This information can also be checked using the Denodo Diagnostic & Monitoring tool in the Data sources tab for JDBC and ODBC sources.



An error raised by Denodo that shows that an outbound connection pool is exhausted is:

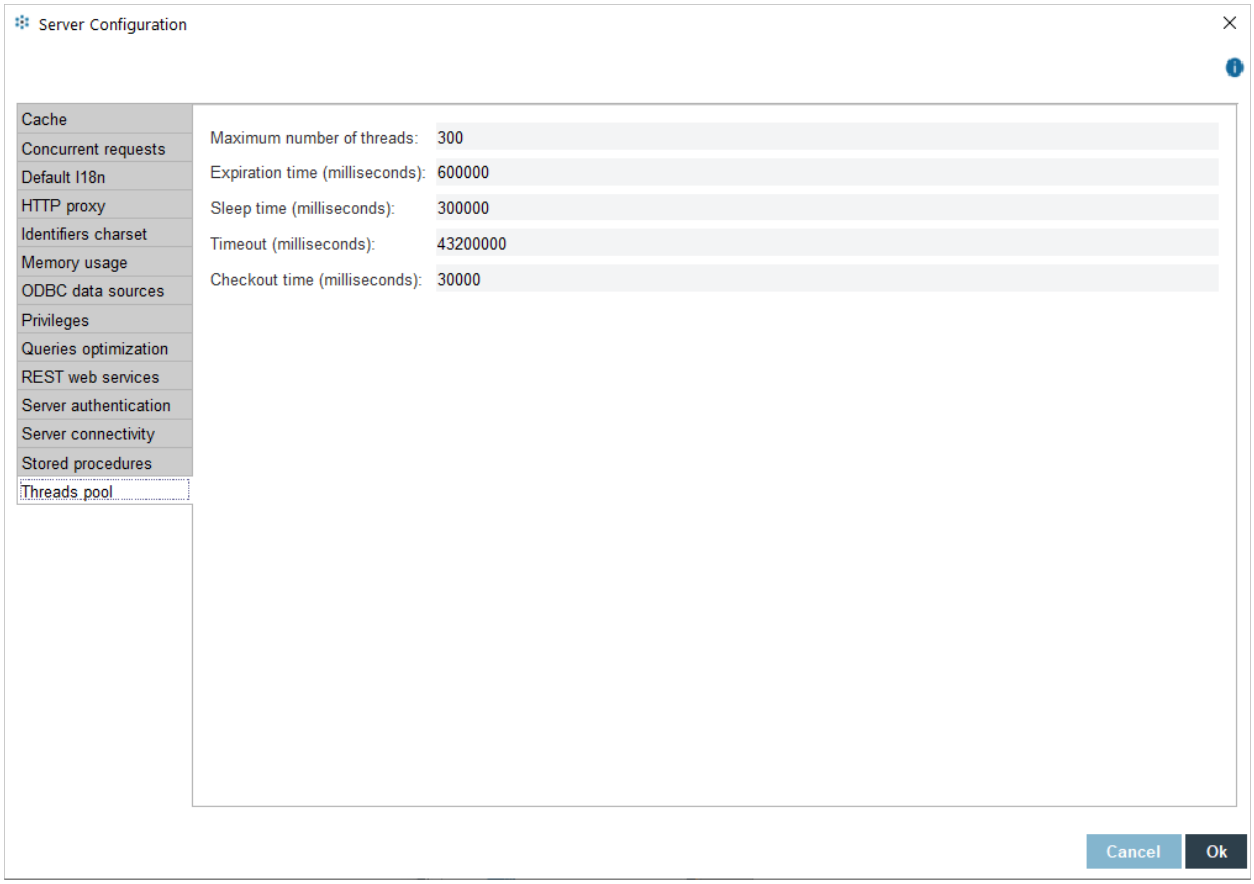
```
 Error getting JDBC connection java.util.NoSuchElementException: Timeout
waiting for idle object
```

5.1.4 Cache Configuration

Denodo's cache is just another JDBC source, so its connection pool should be configured accordingly as well. The cache settings can be modified from the Administration tool menu, `Administration > Server Configuration > Cache`.

5.1.5 Thread Pool

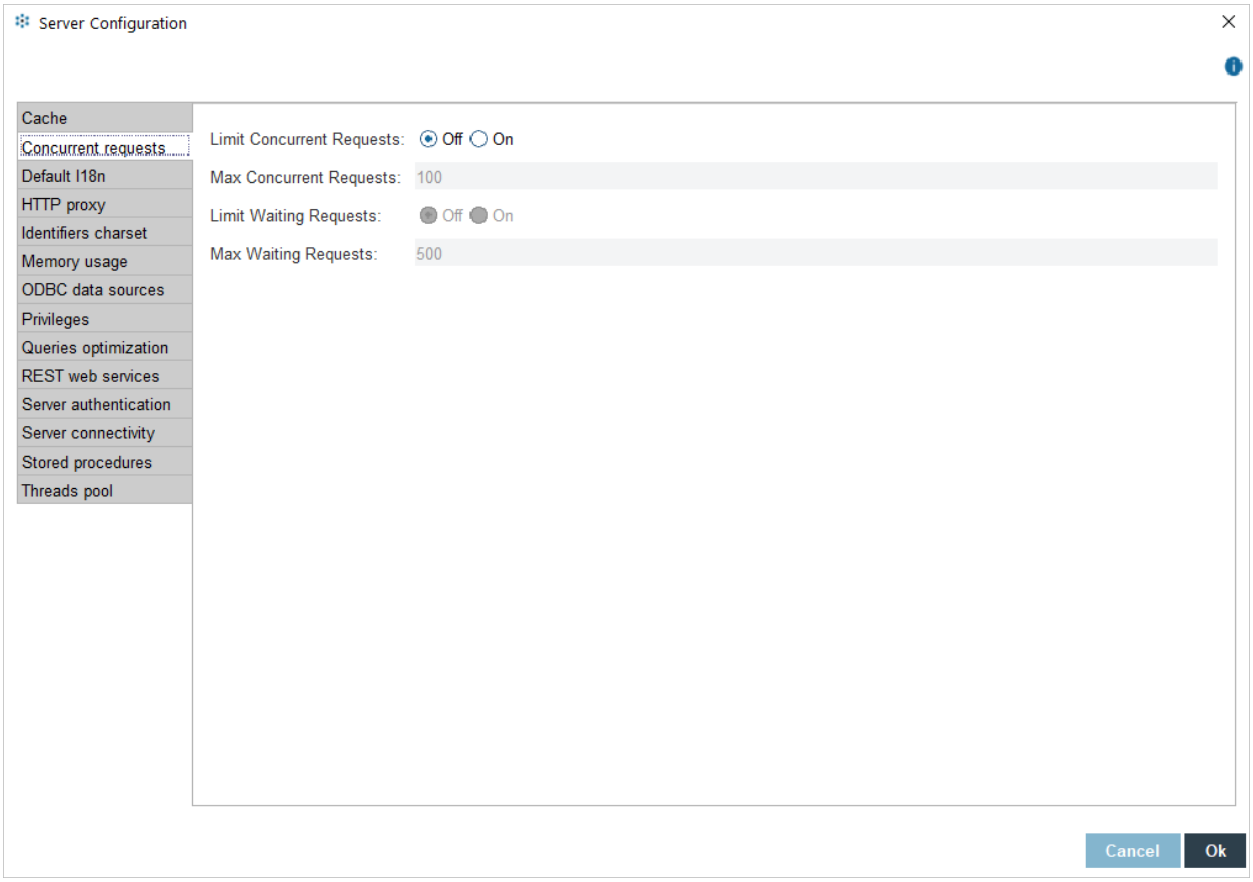
Denodo uses an internal threads pool, which is used by the threads of the execution engine. The parameter "Max threads" will limit the number of threads available for the execution engine. If the number of concurrent requests is too high, then this may lead to exhausting the threads pool. In that case, the max threads parameter should be increased. Other parameters to bear in mind are the thread timeout and checkout time. High concurrency scenarios may slow down the average execution times, producing timeouts in execution or waiting queues. These settings can be modified from the Administration tool menu, `Administration > Server Configuration > Threads Pool`. More information on these parameters can be found in the Virtual DataPort Administration Guide, section "Threads Pool".



The errors raised in Denodo when the threads pool is exhausted will look like:
`com.denodo.vdb.engine.thread.ThreadPool [] - addWork - error requesting thread: Timeout while waiting for an object (checkout)`

5.1.6 Denodo Concurrent request settings

It is possible to limit the number of requests that the Denodo will accept concurrently. When that limit is reached, the new requests will be queued and executed according to their arrival order. Limiting the number of concurrent requests is useful in high load environments, since it avoids performance degradation issues when there is a peak load. These settings can be modified from the Administration tool menu, `Administration > Server Configuration > Concurrent Requests`. For more information on these parameters, please go to the Virtual DataPort Administration Guide, section "Limiting the Number of Concurrent Requests".



5.1.7 Source performance

When Denodo issues a high number of queries to one particular source, the performance of that source can also be affected or even be the main bottleneck of the execution.

5.1.8 Web Container Performance

Web services deployed by Denodo are small client applications running in a web applications container. Denodo's embedded web container is an Apache Tomcat. Considerations on its performance must be taken into account during stress tests as well. Adequate memory configuration and connections pools for the web services with the adequate size and timeouts are important parameters. When the embedded Tomcat is used, it must also be clear that the Apache Tomcat process will compete with

the Denodo VDP server for HW resources, especially CPU and memory.

5.1.9 Web Container Settings

By default, Denodo's embedded web container is configured to handle a maximum of 150 simultaneous threads. If the number of concurrent requests we want to test is bigger than that, this setting must be changed accordingly.

1. Go to `<DENODO_HOME>/resources/apache-tomcat/conf`
2. Edit the following parameter in the `server.xml` file (for Denodo versions 5.5 or older, edit the `server.xml.template` file instead):

```
<!-- Define a non-SSL HTTP/1.1 Connector →
  <Connector port="$TOMCAT_HTTP_PORT"
     maxHttpHeaderSize="8192"
        maxThreads="150"
     minSpareThreads="25"
     maxSpareThreads="75"
       enableLookups="false"
     redirectPort="$TOMCAT_HTTPS_PORT"
     acceptCount="100"
     connectionTimeout="20000"
     disableUploadTimeout="true"
       URIEncoding="UTF-8"/>
```

3. Restart the web container