nirva

# How-to: Performance measurement and optimisation

Document version: 1.01

Nirva can measure performance with the command line tool "nvcc". It can simulate heavy usage of a particular process by defining how many concurrent sessions Nirva can launch and how many loops each session should perform.

It is also possible to display the execution time of each command or procedure on the server.

## Testing performance

This can be done with the nvcc tool. Of course, any other external tool can also be used.

nvcc must be executed from a console window command line with the usual parameters followed by the '-b' option and `l=A:r=B`. B is the number of threads that will execute the command file (please refer to " Tools/nvcc/input file format" in the Nirva documentation for more details) and A is the number of times each thread will execute the command file.

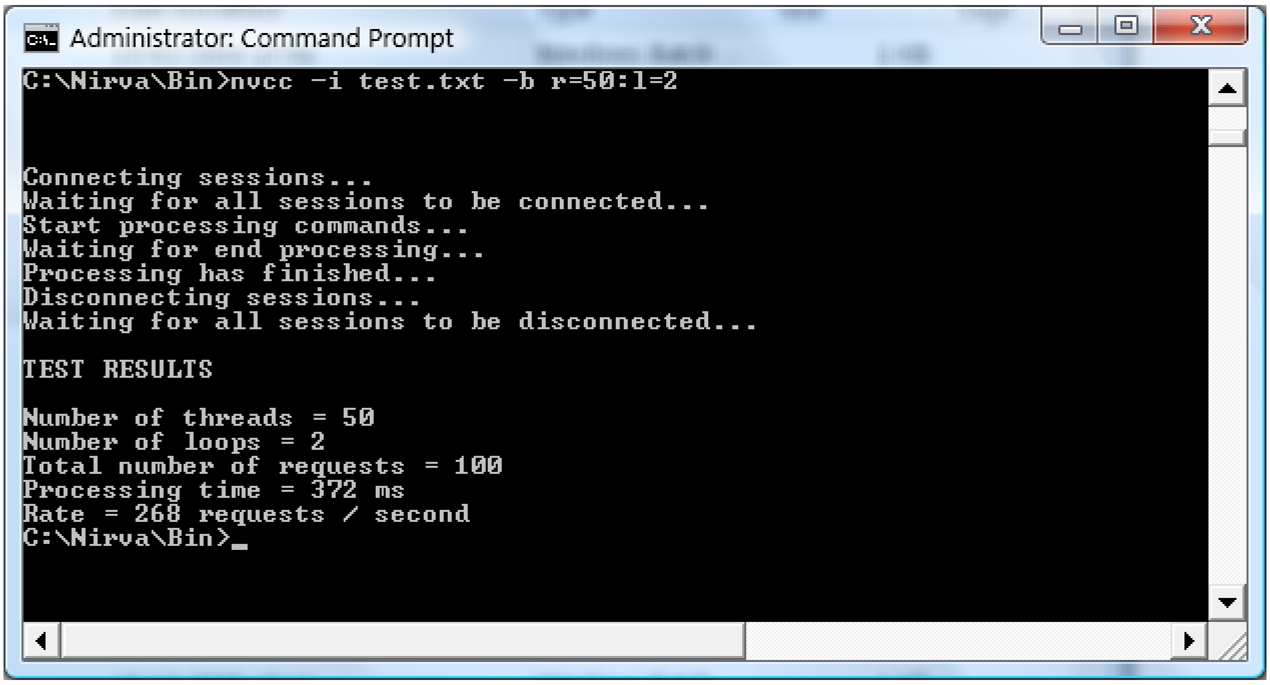
nvcc will then:

- Create the number of threads specified and establish for each of them a connection with the Nirva server, thus creating a new session. Once all the threads are ready, a request is sent to each thread to execute the file specified in the command. The number of threads and the number of executions are specified in the command as described above.
- Wait until all threads have finished the operation and then disconnect them from the server.
- Display the wall clock time needed to execute the processes, not including connecting and disconnecting the sessions.

For all available parameters for nvcc, please refer to the Nirva documentation or execute nvcc with the '-h' option.

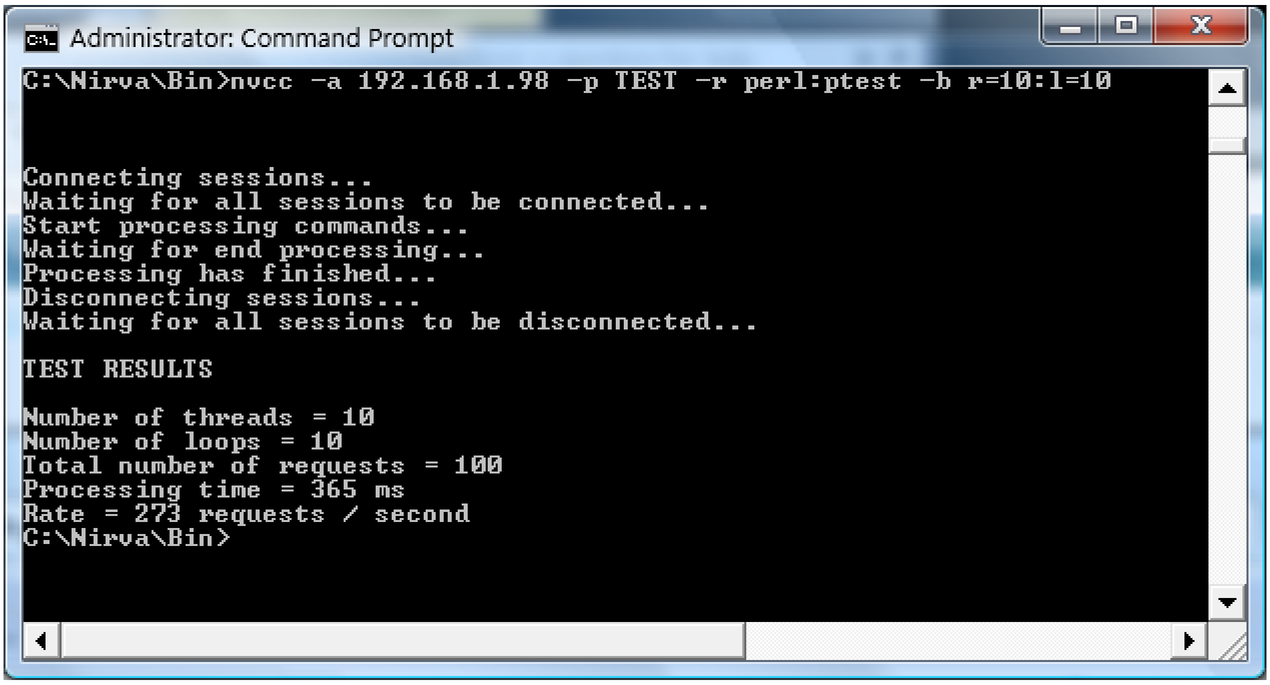
## Example

Please find below an example with a command file on a local Nirva server with the NVDEF application:



```
C:\Nirva\Bin>nvcc -i test.txt -b r=50:l=2


Connecting sessions...
Waiting for all sessions to be connected...
Start processing commands...
Waiting for end processing...
Processing has finished...
Disconnecting sessions...
Waiting for all sessions to be disconnected...

TEST RESULTS

Number of threads = 50
Number of loops = 2
Total number of requests = 100
Processing time = 372 ms
Rate = 268 requests / second
C:\Nirva\Bin>_
```

Example of a direct call of a procedure on a remote server on the TEST application:



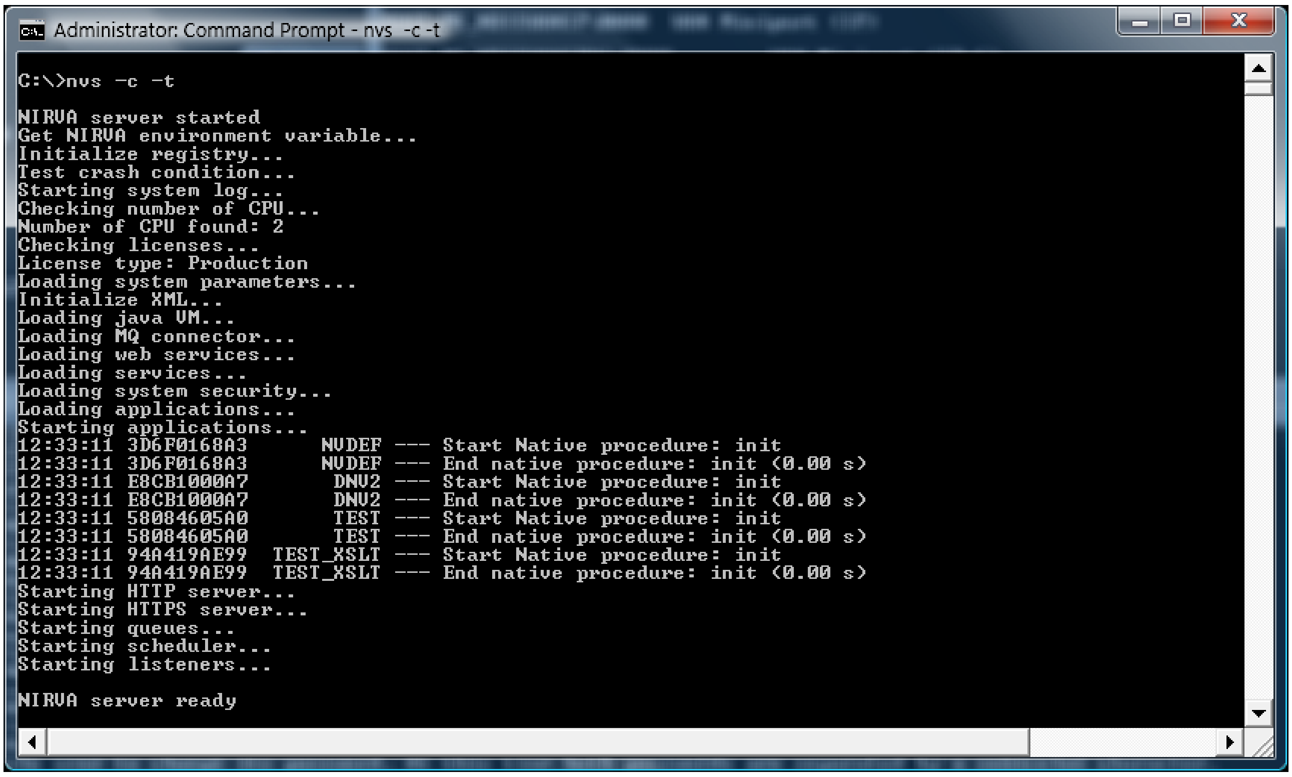
```
C:\Nirva\Bin>nvcc -a 192.168.1.98 -p TEST -r perl:ptest -b r=10:l=10


Connecting sessions...
Waiting for all sessions to be connected...
Start processing commands...
Waiting for end processing...
Processing has finished...
Disconnecting sessions...
Waiting for all sessions to be disconnected...

TEST RESULTS

Number of threads = 10
Number of loops = 10
Total number of requests = 100
Processing time = 365 ms
Rate = 273 requests / second
C:\Nirva\Bin>
```

Performance tests should be done when all unnecessary programs are deactivated. Nirva should not be in debug mode, nor in verbose more and logs must be deactivated (or at least set to an identical level than the production one). Anti-virus systems can significantly impact performance especially when the processes are accessing heavily disk drives.

# Detailed measurement of execution time

Nirva can display execution time of commands or procedures. This function is particularly usefule to diagnose any potential performance issue.

In order to display execution times, Nirva should be executed in console mode with the “-t” option:

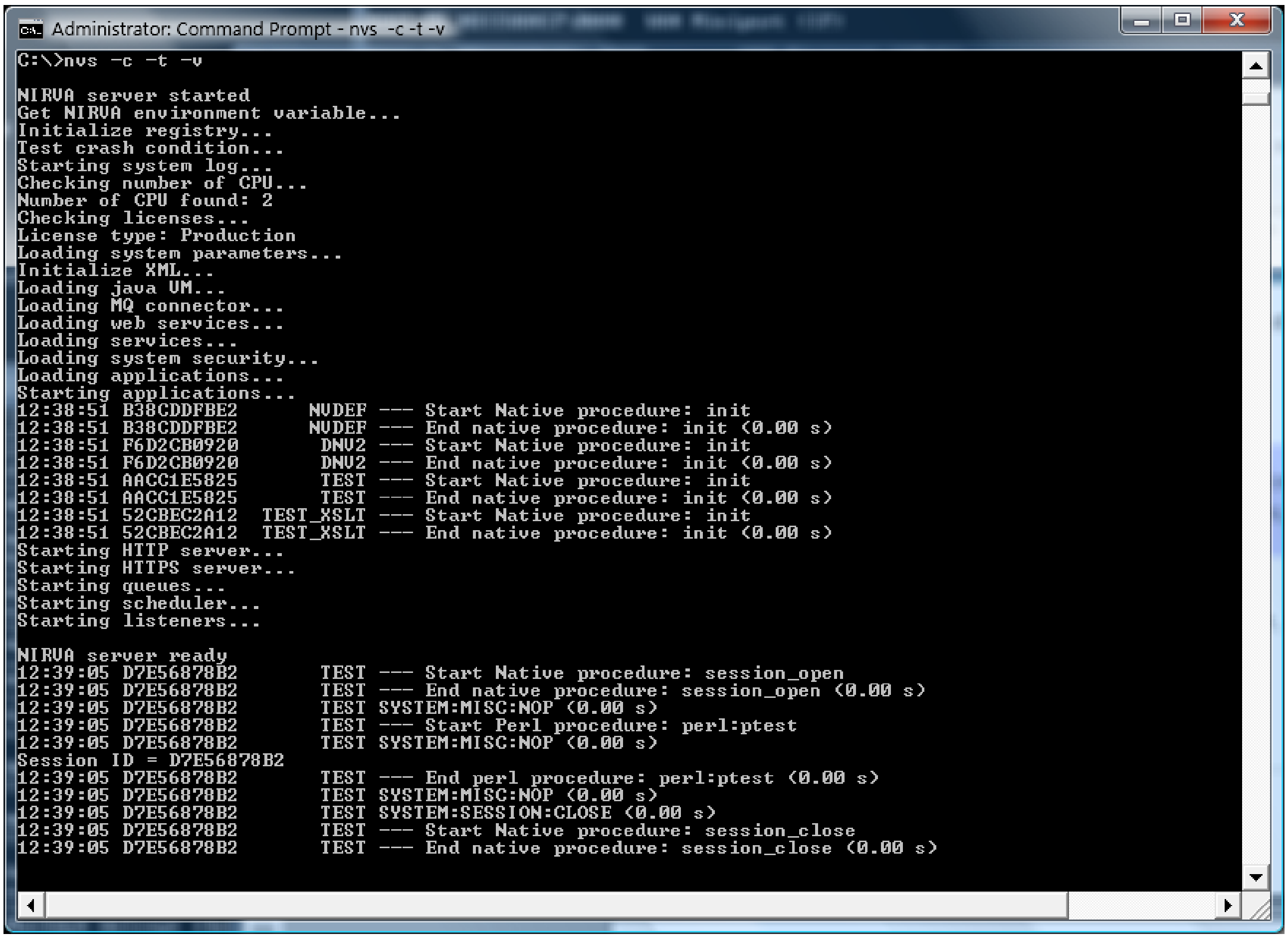


```
C:\>nvs -c -t

NIRVA server started
Get NIRVA environment variable...
Initialize registry...
Test crash condition...
Starting system log...
Checking number of CPU...
Number of CPU found: 2
Checking licenses...
License type: Production
Loading system parameters...
Initialize XML...
Loading java VM...
Loading MQ connector...
Loading web services...
Loading services...
Loading system security...
Loading applications...
Starting applications...
12:33:11 3D6F0168A3      NVDEF --- Start Native procedure: init
12:33:11 3D6F0168A3      NVDEF --- End native procedure: init (0.00 s)
12:33:11 E8CB1000A7       DNV2 --- Start Native procedure: init
12:33:11 E8CB1000A7       DNV2 --- End native procedure: init (0.00 s)
12:33:11 58084605A0       TEST --- Start Native procedure: init
12:33:11 58084605A0       TEST --- End native procedure: init (0.00 s)
12:33:11 94A419AE99  TEST_XSLT --- Start Native procedure: init
12:33:11 94A419AE99  TEST_XSLT --- End native procedure: init (0.00 s)
Starting HTTP server...
Starting HTTPS server...
Starting queues...
Starting scheduler...
Starting listeners...

NIRVA server ready
```

This displays the execution time of external commands and procedures (connectors).

Minimum resolution is 10 milliseconds. A displayed time of 0 indicates that the execution time was less than 10 milliseconds.

In order to display execution time for each command, the "-v" (verbose) option should be used when starting Nirva in console mode:


Administrator: Command Prompt - nvs -c -t -v

```
C:\>nvs -c -t -v

NIRVA server started
Get NIRVA environment variable...
Initialize registry...
Test crash condition...
Starting system log...
Checking number of CPU...
Number of CPU found: 2
Checking licenses...
License type: Production
Loading system parameters...
Initialize XML...
Loading java VM...
Loading MQ connector...
Loading web services...
Loading services...
Loading system security...
Loading applications...
Starting applications...
12:38:51 B38CDDFBE2      NVDEF --- Start Native procedure: init
12:38:51 B38CDDFBE2      NVDEF --- End native procedure: init (0.00 s)
12:38:51 F6D2CB0920       DNV2 --- Start Native procedure: init
12:38:51 F6D2CB0920       DNV2 --- End native procedure: init (0.00 s)
12:38:51 AACC1E5825       TEST --- Start Native procedure: init
12:38:51 AACC1E5825       TEST --- End native procedure: init (0.00 s)
12:38:51 52CBEC2A12  TEST_XSLT --- Start Native procedure: init
12:38:51 52CBEC2A12  TEST_XSLT --- End native procedure: init (0.00 s)
Starting HTTP server...
Starting HTTPS server...
Starting queues...
Starting scheduler...
Starting listeners...

NIRVA server ready
12:39:05 D7E56878B2       TEST --- Start Native procedure: session_open
12:39:05 D7E56878B2       TEST --- End native procedure: session_open (0.00 s)
12:39:05 D7E56878B2       TEST SYSTEM:MISC:NOP (0.00 s)
12:39:05 D7E56878B2       TEST --- Start Perl procedure: perl:ptest
12:39:05 D7E56878B2       TEST SYSTEM:MISC:NOP (0.00 s)
Session ID = D7E56878B2
12:39:05 D7E56878B2       TEST --- End perl procedure: perl:ptest (0.00 s)
12:39:05 D7E56878B2       TEST SYSTEM:MISC:NOP (0.00 s)
12:39:05 D7E56878B2       TEST SYSTEM:SESSION:CLOSE (0.00 s)
12:39:05 D7E56878B2       TEST --- Start Native procedure: session_close
12:39:05 D7E56878B2       TEST --- End native procedure: session_close (0.00 s)
```

## Performance optimisation

It is possible to use load balancing functions to optimise the performance of Nirva enabled processes by adding more servers and running processes in parallel. To this end, the Nirva Workflow service greatly simplifies the setup of business applications with high performance in a multi-server architecture.

It is also possible to develop procedures taking performance into account. A Nirva command is relatively fast but needs decoding parameters. Other operations can take a certain time as well. Whenever possible, Nirva commands should be sparingly used, especially in loops. Nirva supplies object manipulation commands that allow working on a number of entities with a single command.

Here is an example of a table with not optimised commands:

```
NV_CMD=|OBJECT:CREATE| NAME=|TEST| TYPE=|TABLE|
NV_CMD=|OBJECT: TABLE_INSERT_COLUMN| NAME=|TEST| COLNAME=|Name|");
NV_CMD=|OBJECT: TABLE_INSERT_COLUMN| NAME=|TEST| COLNAME=|Items|");
NV_CMD=|OBJECT: TABLE_INSERT_ROWS| NAME=|TEST|");
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|0| COLNAME=|Name|
LINE=|Name1|");
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|0| COLNAME=|Items|
LINE=|Item11|");
```

```
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|0| COLNAME=|Items|
LINE=|Item12|");
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|0| COLNAME=|Items|
LINE=|Item13|");
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|1| COLNAME=|Name|
LINE=|Name2|");
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|1| COLNAME=|Items|
LINE=|Item21|");
NV_CMD=|OBJECT:TABLE_INSERT_CELL_LINE| NAME=|TEST| ROW=|1| COLNAME=|Items|
LINE=|Item22|");
```

The code above uses 11 Nirva commands to create and populate a relatively simple table.

Here is the same functionality developed with a single command:

```
NV_CMD=|OBJECT:CREATE| NAME=|TEST| TYPE=|TABLE| COLUMNS=|Name;Items| ROWSEP=|§|
LINESEP=|µ| COLSEP=|;| VALUE=|Name1;Item11µItem12µItem13§Name2;Item21µItem22|
```

This simple example illustrates how certain command scan be used more effectively to optimise code and therefore increase performance. Please find below a non exhaustive list of some commands of the OBJECT class that could help increasing performance:

- CREATE
- STRINGLIST_GET_VALUES
- STRINGLIST_INSERT
- STRING_LIST_SET_VALUE
- INDSTRINGLIST_GET_VALUES
- TABLE_ADD_COLUMNS
- TABLE_ADD_ROWS
- TABLE_CLEAR_ROWS
- TABLE_GET_COLUMN
- TABLE_GET_ROW
- TABLE_GET_ROWS
- TABLE_INSERT_ROWS
- TABLE_IMPORT
- TABLE_EXPORT
- TABLE_SEARCH
- TABLE_SET_ROW