

CESNET Technical Report 28/2004

Scheduling and processing of performance monitoring tests

Antonín Král <A.Kral@sh.cvut.cz>, Czech Technical University
Sven Ubik, <ubik@cesnet.cz>, CESNET, Prague, Czech Republic

December 13, 2004

Abstract

In this report we present a simple framework for scheduling various kinds of performance tests, processing and presenting their results. This framework allows us to experiment with various performance monitoring tools and observe their behaviour when used simultaneously between specified points in our network.

Keywords: performance monitoring, test scheduling,

1 Performance monitoring

When data traffic is being transferred over the network it experiences various performance characteristics, such as throughput, delay, packet loss rate, jitter, etc. A lot of performance monitoring tools have been developed to measure these performance characteristics or to check the current state of the network in order to verify that the required performance characteristics can be achieved. Performance monitoring tools are indispensable for locating fault points, performance affecting points and to observe trends in the network operation.

Different characteristics require different measurement methods. Also different uses require alternative result processing and presentation. Tests can be started on-demand or scheduled regularly. Measured values need to be appropriately aggregated. Consequently, there are many different tools available, see for instance [1] or [2]. For a comprehensive view on various performance characteristics we need to use a set of tools. This implies a need for an extensible framework for scheduling individual tests and for processing and presentation of their results.

As part of our participation in JRA1 activity (Performance measurement and management) of the GN2 project [3] developing the GÉANT2 network we developed a pilot version of the test scheduling and result processing framework. It allows us to acquire experience with various tools and observe their behaviour when used simultaneously between specified points in our network.

2 Test specification

Information about what tests should be run as well as measured and processed results are all stored in MySQL database with the structure illustrated in Fig. 1.

The purpose of each table is as follows:

test_type Includes one record for each test type (e.g., iperf for throughput measurement or owamp for one-way delay measurement).

tests Includes one record for each instance of each test type, one test type can run between several different end points.

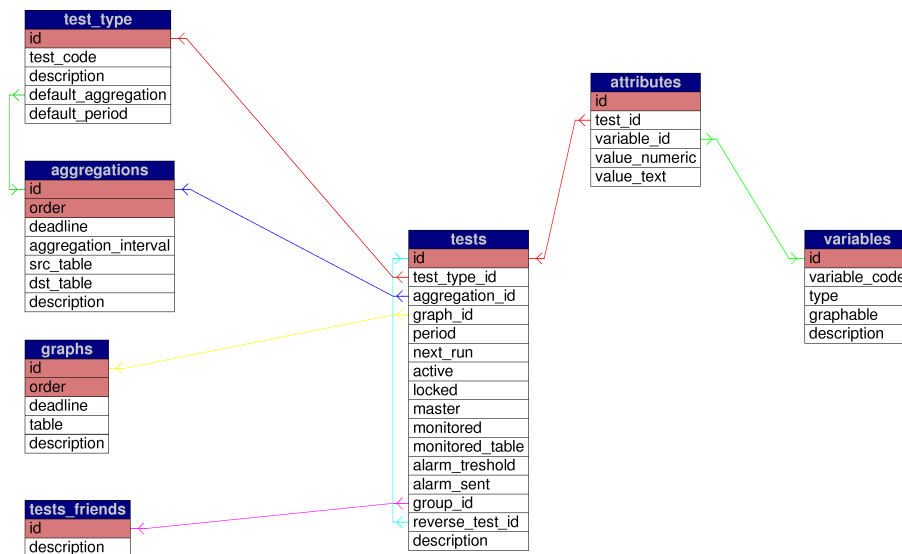


Figure 1: Database structure

attributes Includes parameters of test instances, such as IP addresses of end points.

mvalues Includes (raw, unprocessed) measured results.

variables Describes the meaning of each value in **attributes** and **mvalues** tables.

aggregations Describes how measured values in **mvalues** table should be aggregated.

mvaluesxxx These tables (several tables with the common prefix of **mvalues**) include aggregated results from **mvalues** table according to the specification in **aggregations** table. The aggregation is performed by Perl scripts.

graphs Describes what graphs should be produced.

tests_friends Describes what test instances belong to one group. Results of all test instances in one group can be presented in the same set of graphs for easy comparison. Also, you can specify that a certain subset of tests in a group should be executed in sequence and not in parallel when parallel execution would affect measurement results (e.g., parallel iperfs from one end point).

Test specification is currently set up by direct editing of corresponding tables in the MySQL database. We use a PHP MyAdmin front-end for a web-based user interface to the MySQL database. For instance, editing **test_type** table is illustrated in Fig. 2.

3 Data aggregation

Aggregation is performed by Perl script **avg-data2.pl**. This script should be run periodically approximately each 10 minutes from the cron daemon or from a shell script like *while true ; avg-data2.pl ; sleep 500 ; done*. There is no link between data aggregation intervals and the interval in which **avg-data2.pl** script is run. Aggregation intervals are specified by **aggregation_interval** field in **aggregations** table.

The **avg-data2.pl** script includes two main subroutines – **performAggregation** and **performTableTrim**.

The **performAggregation** subroutine computes the range of timestamps over which the data should be aggregated and performs aggregation according to the test type. Therefore, this subroutine is test-type dependant and should be extended when a new test type is added. Data

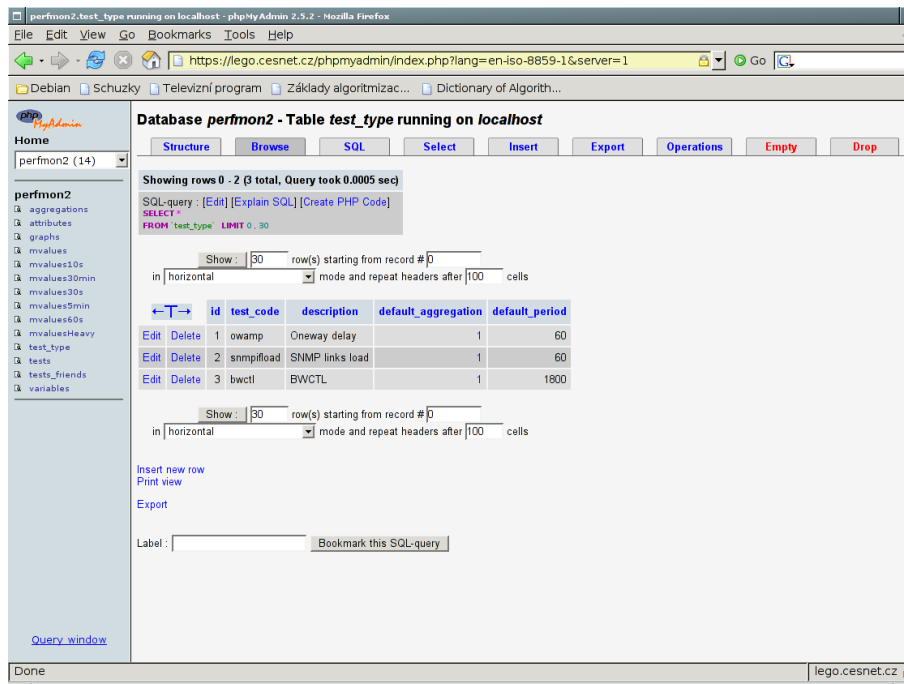


Figure 2: Editing `test_type` table

is read from the table specified by `src_table` field and written to the table specified by `dst_table` field.

The `performTableTrim` subroutine deletes all data records from the table specified by `src_table` field which have timestamps older than the current timestamp minus `deadline` field. If `deadline` is 0 (zero), no data is removed and it stays in the table forever.

4 Test scheduling

The test scheduler is implemented by Perl script `scheduler.pl` and needs to be started manually at the beginning of monitoring. It then acts according to the following algorithm based on the content of `tests` table:

1. Get minimum of `next_run` field and sleep up to this time or for the maximum of 20 seconds (to check for possible changes in `tests` table).
2. Select all test instances which have `next_run` \leq the current timestamp. Order these tests by `next_run` and then `id`. Set `locked` to 1 for these tests.
3. For all selected tests run their child processes to perform the actual tests. This is another place which must be modified when a new test type is added. See section 5.
4. After each selected test is performed, schedule the next test run as:

$$\text{next_run} = \text{next_run} + ((1 + \text{int}(\text{current_time} - \text{next_run}) / \text{period})) * \text{period}$$

That is if one or more periodic intervals of test executions passed while the original execution took place, these intervals will be skipped and the next execution will be at the next interval.

5. After scheduling the next test run, set its `locked` back to 0.

If there are more tests in one group (they have the same value of `group_id` field), the tests with `master` set to 1 are executed first in parallel (they are started in sequence according to their `id` value). After all tests with `master` set to 1 finish, the remaining tests in the group (with `master` set to 0) are executed in sequence according to their `id` value, that is the next test is started when the previous test finished.

Execution of tests at remote sites is done by NRPE [4]. A remote site executes a simple Perl script which wraps particular test command (e.g., `owamp`) and translates its output to simple one line format, which is processed by the child process started from the scheduler and written to the database.

5 Adding new tests

Adding a new instance of existing test type is simple. Add its parameters to `attributes` table, insert a new line to `tests` table and refer appropriate lines in `aggregations` and `graphs` tables.

Adding a new test type is more complex. Perform the following steps.

1. Insert a line to `test_types` table and if needed add description of new test parameters to `variables` table. You will probably also need to add lines to `aggregations` and `graphs` tables.
2. Add support for the new test type to scheduler. Look at function `executeTest` in `scheduler.pl` script and find the part starting with `FORK:`. You will find something like this:

```
# child here
if ($testtype == 1){
    # test type == owamp
    executeOWAMP($testid, \%testparams);
}
```

Add new `if` statement for the new test type and call your new subroutine which you will create in next step.

3. Create a new subroutine for executing the new test type in a similar fashion as `executeOWAMP` subroutine. You will see that after getting input values from the database, NRPE is executed with some parameters and then output from NRPE (which transmits stdout from your command in the remote site, probably parsed through some wrapper script) is parsed.
4. Configure NRPE in `nrpe.cfg` file. Add a line similar to this (everything is on one line):

```
command[owping]=/home/perfmon/perfmon/distr/bin/owping $ARG1$ $ARG2$
| /home/perfmon/perfmon/distr/bin/owping-wrapper.perl
```

5. Extend `performAgregation` subroutine in `avg-data2.pl` script to perform aggregation for the new test type in a similar fashion as it is done for existing test types.
6. Extend function `printTestTypeSelection` in `graphs2.cgi` script. If you want a customized graph legend, take a look at `make_graph` function. Find the part marked with comment `# Graph legend` and add your test-type specific code.

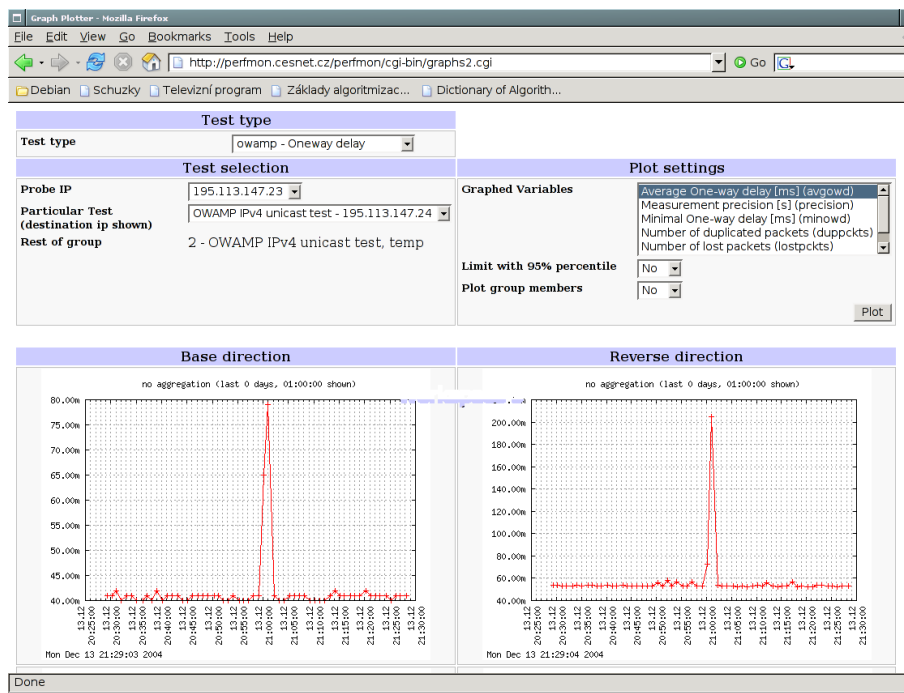


Figure 3: Result presentation user interface



Figure 4: Enlarged graph of throughput measured by bwctl

6 Test result presentation

All measured results from all tests are available in a uniform web-based user interface illustrated in Fig. 3. The user can click on any graph to display it in a larger resolution as illustrated in Fig. 4. Every test instance can have a set of graphs associated with it. `graph_id` field in `tests` table references `id` field of `graphs` table. Several rows in `graphs` table may have the same value of `id` field, which forms a table key together with `order` field. Data is read from table specified by `table` field. It can be `mvalues` for measured (non-aggregated) data or one of `mvaluesxxx` tables for aggregated data. Each graph will display values for the range of timestamps from the current timestamp minus `deadline` field up to the current timestamp. If `deadline` is 0 (zero) the whole table is drawn.

The user first selects the type of test. Currently we use three types of tests - `bwctl` (wrapper around `iperf`) for active throughput measurement, `owamp` for active one-way delay measurement and reading router interface byte counters via `SNMP` for instantaneous link load measurements.

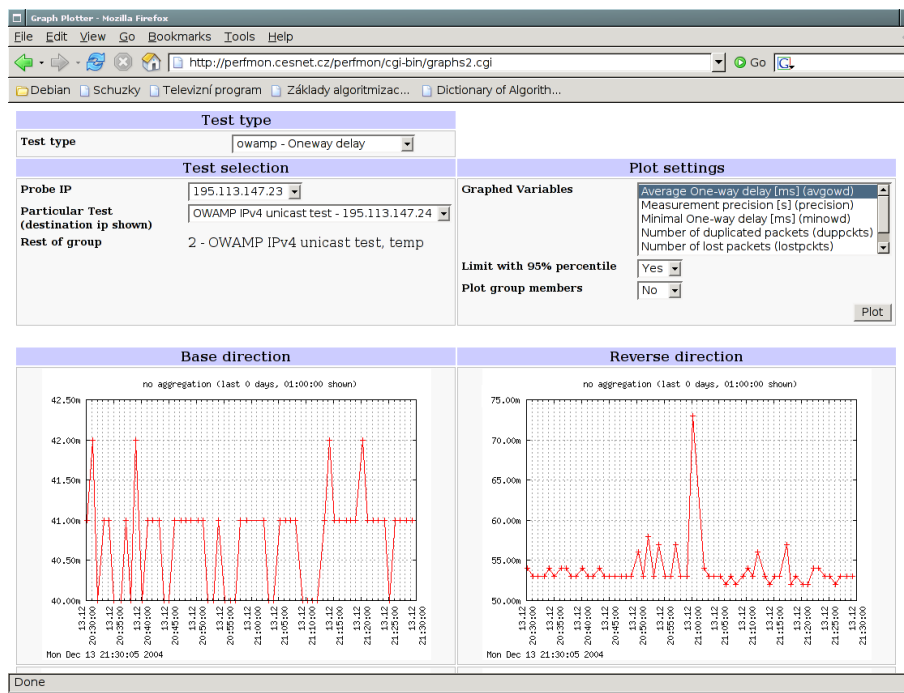


Figure 5: Omitting values exceeding 95% percentile

We plan to add further tests as needed. The system automatically finds all test instances of the selected type.

In the next step the user selects a particular test instance by specifying its end points. The system automatically finds all performance characteristics measured by this test instance.

After that the user chooses one or more performance characteristics to be plotted in graphs. If more performance characteristics are selected, each of them is plotted in a different colour and line style.

Next, the user can choose if values exceeding the 95% percentile should be omitted from graphs. This allows to utilize the graph space to display most values in final detail where otherwise a few excess values would compress most values to a small portion of the graph space. Fig. 5 illustrates presentation of the same measurements as in Fig. 3, but with values exceeding the 95% percentile omitted.

Finally the user may request to plot results of all other test instances which belong to the same group as the selected instance together in one set of graphs for easy comparison. An example graph of throughput measured by bwctl from the selected end point to several other end points is illustrated in Fig. 6.

Processed results, that is date which is aggregated in several different timescales, are automatically presented under the measured (unaggregated) results, as shown in Fig. 7.

7 Conclusion

We presented a simple framework for scheduling various kinds of performance tests, processing and presenting their results. For the purposes of inter-domain measurements in the Géant2 network we will need to integrate our measurements with measurements from other NRENs and Géant2 backbone.

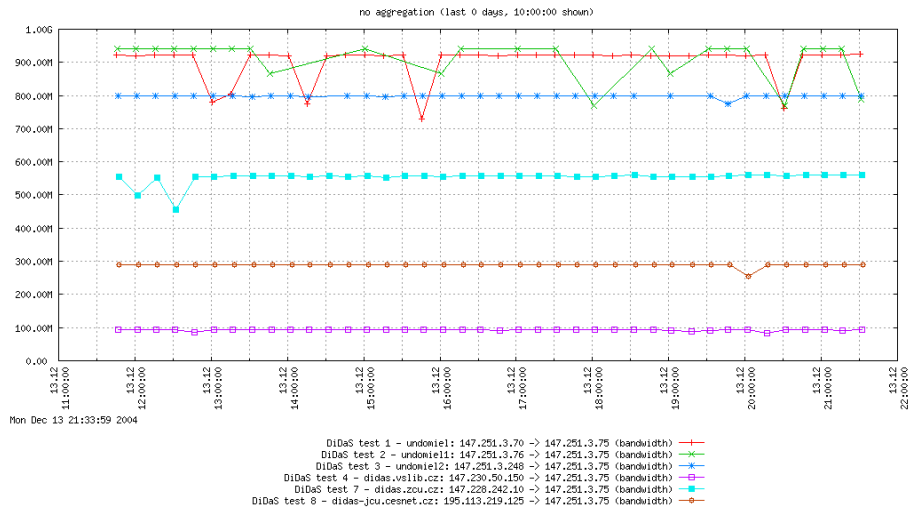


Figure 6: Plotting a group of test instances

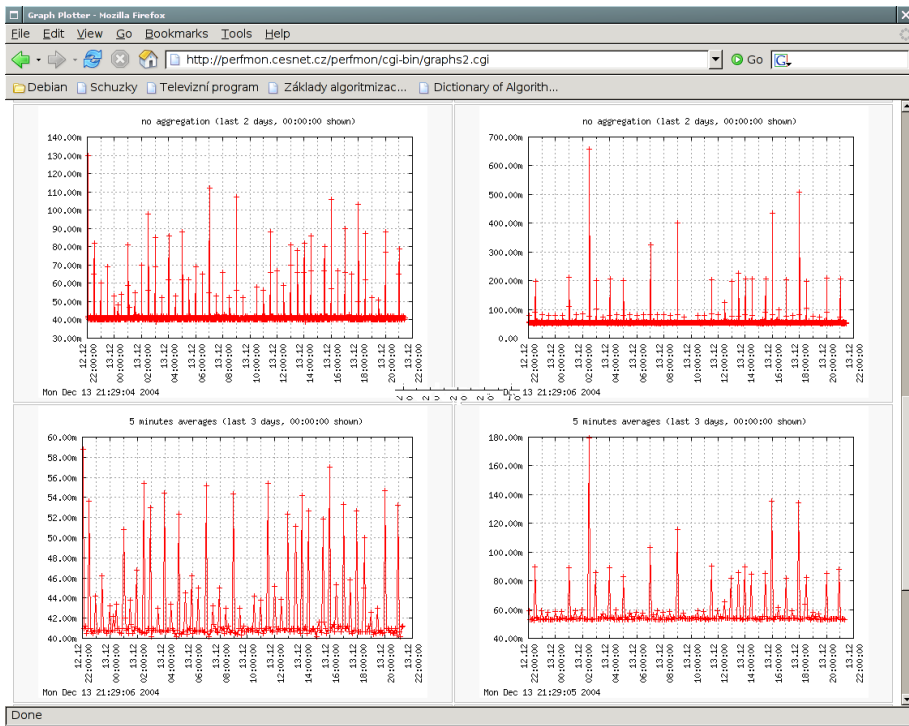


Figure 7: Processed (aggregated) results

References

- [1] Cooperative Association for Internet Data Analysis (CAIDA), <http://www.caida.org>.
- [2] Internet End-to-end Performance Monitoring (IEPM), <http://www-iepm.slac.stanford.edu>.
- [3] GÉANT2, <http://www.geant2.net>.
- [4] <http://www.nagios.org/download/extras.php> and look for NRPE.