Poster: NPF: orchestrate and reproduce network experiments

Tom Barbette
UCLouvain
Belgium

ABSTRACT

Networked systems researchers face challenges in their development process, transitioning from local testing to scaling experiments on public testbeds. The manual creation of ad-hoc scripts for parameter exploration and analysis leads to inefficiencies, hindering the development process and reproducibility.

In this poster, we present a Network Performance Framework (NPF) that orchestrates experiments automatically, from the step where a prototype is ready for its first run to the final graphs one can embed in a paper. The framework generates graphs for each metric, interactive web pages and Jupyter notebooks that allow users to interact easily with the experiment's results.

NPF is easy to start with, as the simplest test description file is just a list of bash commands to run. Gradually, NPF makes it easy to deploy an experiment over local and remote testbeds, enabling complex deployment, synchronization, and data collection. Each experiment can grow a set of factors (e.g., number of threads, buffer size, packet length, packet rate, . . .) that NPF uses to orchestrate the experiment and collect performance metrics (e.g., throughput, latency, . . .) of different configurations. NPF helps the researcher in building their experimental design to select meaningful factors, automatically finding regions of interest and easily cutting through multi-dimensional data.

1 NETWORKED SYSTEMS RESEARCH

Typical networked system research involves developing a prototype that will be first functionally evaluated with a client and a server running on a researcher's laptop. Progressively, more realistic experiments will be conducted over real hardware. For many systems researchers, a local testbed will be needed to use some exotic devices (SmartNIC, P4 switches, ...), or need dedicated machines during a long development process. Some experiments are then conducted at large scale over public testbeds with reservation systems like Grid5K[4].

In parallel, the following research cycle will be repeated many times and on each testbed scale. A new feature or new factor will be selected to study the system under test and launch a novel experiment. The researcher will then dig into the generated data, often through visualization, to evaluate the pertinence of the new angle or new feature.

In this poster, we present **NPF**, a tool designed to assist researchers in navigating between testbeds and repeating this research cycle. NPF streamlines the process by providing a unified experiment description file capable of accommodating various testbed scales and numerous parameter sets. It will enforce good standard practices like repeating a test multiple times and reporting the variance. As the number of parameters that could impact a given system quickly grows exponentially, NPF proposes multiple tools

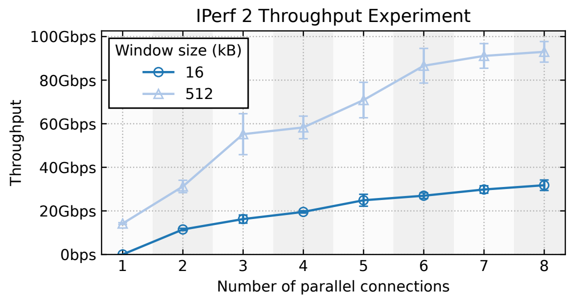
```
%info
IPerf 2 test
This tests measures the throughput of client/server TCP connection(s)
%variables
PARALLEL=[1-8]
WINDOW={16,512}
%config
var_names={PARALLEL:Number of parallel connexions,
           WINDOW:TCP Window size (kB)}
%script@server
iperf -s
%script@client
iperf -c ${server:0:ip} -P $PARALLEL -w $WINDOWk &> log
echo "RESULT-THROUGHPUT $(cat log | grep -ioE '[0-9.]+ [kmg]bits')"
```

(a) The NPF script file.

```
npf-run --test iperf.npf --cluster client=gros-95.nancy.grid5000.fr
                                   server=gros-96.nancy.grid5000.fr

...
PARALLEL = 8, WINDOW = 16 [run 4/5 for test 8/16]
gros-95 - server [0] ------------------------------------------------------------
gros-95 - server [0] Server listening on TCP port 5001
gros-95 - server [0] TCP window size:  128 KByte (default)
gros-95 - server [0] ------------------------------------------------------------
gros-95 - server [0] [  1] local 10.221.0.1 port 5001 connected with 10.221.0.2 port 47866
gros-95 - server [0] [ ID] Interval       Transfer     Bandwidth
gros-95 - server [0] [  1] 0.0000-2.0002 sec  1.16 GBytes  4.98 Gbits/sec
gros-96 - client [0] RESULT-THROUGHPUT 4.98 Gbits
...
```

(b) A sample command line to run the script and its partial output



(c) Graph automatically generated on our local 100G machines.

Figure 1: Simple experiment using iPerf2

to easily go over many areas of the experimental space and quickly go through the results, finding why the system behaves differently than expected. NPF also enables comparing different systems under the same parameters. We believe NPF offers a unique, easier chain than existing systems to define and evaluate an experimental design. NPF also provides useful tooling like performance regression testing not covered here.

NPF, however, does not try to tackle the matter of deploying a physical topology or loading operating system images on nodes. Researchers might use Mininet[2] to deploy a virtual topology on a local machine, use a local testbed that is not automatable or deploy the same experiment over a real large-scale topology on Grid5K with an Enoslib[3] script. NPF uses SSH to deploy and execute an experiment over those physical or virtual platforms, with capabilities for setup and teardown of the experiment. Therefore, NPF does not suffer from being tied to specific platforms like Enoslib which does not offer an easy way to deploy an experiment on a local testbed, or POS[5] that is tied to its author's testbed. This is a strong limitation for researchers who need to use local hardware or multiple testbeds.

2 NPF

Figure 1 (a) shows a simple NPF script that will run iPerf2 between a client and a server to evaluate the performance of one or more TCP connections. The heart of the experiment is based on bash scripts, as we expect the researcher to start running their own external program with a few various parameters and iteratively take advantage of NPF to generate complex configuration files and deployment scenarios. The command to launch the experiment is shown in figure 1 (b), along with one of the replications (4/5) of one of the 16 possible variable combinations. The resulting graph as produced by NPF can be seen in figure 1 (c). NPF itself is written in around 7K lines of Python 3.

Deployment NPF will handle a series of scripts to run on machines taking a specific role. In the example, there are two roles: client and server. Roles are, in turn, mapped through the command line to physical nodes according to the desired topology. Figure 1 shows the IP address of the server is taken from its actual configuration, with possible tweaking falling outside the scope of this short introduction. NPF can ensure synchronization between multiple scripts, ensuring one command runs after some initialization phase has finished, for instance. The NPF script can also generate application configuration files using a templating system, which will then be distributed to the relevant roles over ssh.

Experimental design NPF will re-execute the experiment under various different settings as described by the factors defined in the `%variables` section of the test script. NPF supports expressing many types of experiment variables, such as a list of parameters, ranges, and ways to explore them, such as logarithmic exponentiation or WSP[1] to explore a subset of the space maximizing distance in the parameters space.

Re-useable modules NPF comes with the support of modules, a way to define standard sub-scripts following the same format. Modules can spawn a packet generator, run a profiling tool like Perf to count the number of CPU events or build a graph of the time spent in the system's functions, a standard nginx web server on a particular role, an HTTP client generator, or ensure a particular CPU frequency.

2.1 Data collection

The data collection process is based on the standard output, as shown in the example. The script can report many metrics that will be collected. NPF will parse the standard output of all scripts for the particular format `RESULT-METRIC X`. It also supports metrics evolving over the course of the experiment like a latency metric reported every second.

2.2 Analysis of results

NPF can export the data collected as CSV but provides multiple internal tools to quickly analyse the result of an experiment.

Graphing A graph is automatically generated for each metric of the experiment, using different types of plots (supporting heatmaps, boxplots, bar plots, scatter plots, error bars, ...), grouping factors when needed to make the results understandable as quickly as possible. While providing some styling options for a publication-ready graph, the primary goal is to provide the researcher with a simple visualization during the development phase.

Statistical analysis As the number of factors can quickly grow, NPF will compute the feature importance of each factor, along with other statistics and metrics of covariance to help the researcher decide if some variant of the software under test provides better performance. After assessing if a factor is only marginally impacting results, the researcher can inspect in more detail only those factors of importance. NPF includes a database so the results of an experiment are cached, only executing the experiment for the new value of factors.

Web page NPF can generate a web page that is more interactive than a graph, changing which factor is used for every axis and splitting a graph in multiple sub-graphs, one per value of a factor. The web page generated for figure 1 is available at https://tbarbette.github.io/npf-web-sample/. This can be used to link an interactive online version for the equivalent graphs of a paper.

Jupyter notebook NPF can also generate a notebook, including the data collected from the experiment and code to generate appropriate graphs, ready to be tempered.

3 CONCLUSION

The goal of this poster is to invite researchers to try our tool which we believe fills a space to enable more reproducible research. At the same time, NPF greatly helps its users to develop large-scale experiments in no time. Experience in artifact evaluation committees also shows having an ecosystem of a few frameworks that, like NPF, fill that space for the specific needs of their community would greatly help in understanding the canvas and reproducing experiments.

NPF is available at https://github.com/tbarbette/npf and the documentation on http://npf.readthedocs.io. It is distributed under the GPL3 license.

4 ACKNOWLEDGEMENTS

The author would like to acknowledge all contributors of NPF, particularly Simon Cornelis for developing the Jupyter Notebook integration and Philippe Dan for the website generation.

REFERENCES

[1] J. Santiago et al. 2012. Construction of space-filling designs using WSP algorithm for high dimensional spaces. *Chemometrics and Intelligent Laboratory Systems* 113 (2012), 26–31.

[2] K Karamjeet et al. 2014. Mininet as software defined networking testing platform. In *International conference on communication, computing & systems (ICCCS)*. IEEE, 139–42.

[3] R-A. Cherrueau et al. 2021. Enoslib: A library for experiment-driven research in distributed computing. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (2021), 1464–1477.

[4] R. Bolze et al. 2006. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications* 20, 4 (2006), 481–494.

[5] S. Gallenmüller et al. 2021. The pos framework: A methodology and toolchain for reproducible network experiments. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 259–266.