Excalibur-Neptune report 2047356-TN-10-1

Task 0.2

Development of a performance testing and aggregation tool for NEPTUNE proxy-apps

Edward Higgins and David Dickinson

University of York

April 1, 2022

Contents

1 Executive Summary 1

2 Motivation & Requirements 2
- 2.1 Motivation 2
- 2.2 Requirements 2

3 Existing technology 3
- 3.1 Automation 3
  - 3.1.1 GitHub Actions and self-hosted runners 3
  - 3.1.2 ReFrame HPC 3
- 3.2 Performance analysis 4
- 3.3 Data aggregation and visualisation 4

4 Design & Implementation 4
- 4.1 Overview 4
- 4.2 RoundTable backend 5
  - 4.2.1 Proxy app configuration 5
  - 4.2.2 Benchmark scripts 6
  - 4.2.3 Database upload 7
- 4.3 Database 7
- 4.4 RoundTable dashboard 7
- 4.5 Availability 8

5 References 8

1 Executive Summary

This report outlines the requirements and design of a software framework for the automated performance testing and performance regression analysis of git-managed software, for example the proxy apps developed as part of the ExCALIBUR NEPTUNE project.

The framework allows acceptance tests and performance benchmarks to be run automatically using GitHub Actions on self-hosted runners at a range of local and HPC facilities such as Bede or Archer2. The results from these runs are then collated and displayed in a web dashboard along with results from other applications, sites and code versions.

The motivation of the project is both to allow easy tracking of performance improvements during the development of the proxy apps, as well as to identify if and when any code changes negatively affect the performance of such applications across a range of hardware and compiler combinations.

2 Motivation & Requirements

2.1 Motivation

The aim of the ExCALIBUR NEPTUNE project is to develop software and infrastructure for simulating the behaviour of tokamak plasma edges, one of the grand challenges[1]. In order to do this, software solutions will need to be developed using a range of algorithms across a number of different software and hardware configurations, and the real-time performance and scaling of these solutions monitored.

It is often said that no significant software application is perfectly bug-free, and the same is true in terms of optimal performance. Often, changes in one part of the code can negatively affect application performance elsewhere in unexpected ways. Being able to spot this performance decrease quickly, as well as identifying which particular code change caused the decrease is incredibly useful for rapid development of highly performant software.

Similarly, code that performs well on one system may not necessarily perform well on others. This can be due to differing compiler/library availability, or differences in the underlying hardware performing the calculations. For example, algorithms that work well on GPUs are often different to those that work well on CPUs, due to their focus on massive parallelisation rather than single-threaded performance. Being able to compare the performance of software across a range of hardware allows these differences to be understood.

2.2 Requirements

Due to these considerations, a framework for automated performance testing and analysis of the NEPTUNE proxy apps was designed. The requirements for the framework were:

1. Automated deployment: The framework needs to be able to build, run and analyse the performance of proxy apps automatically when triggered;
2. Flexibility: The framework needs to be able to:
   (a) Compile and run the range of software seen in the proxy apps (e.g. Fortran/C, MPI, CUDA etc.)
   (b) Run on the range of hardware systems & software environments of interest (e.g. x86/ARM CPUs, GPUs etc.)

   (c) Obtain a range of performance metrics. (e.g. run time, parallel efficiency, memory usage etc.)

3. Interpretability: The framework needs to be able to display the results from these runs in such a way that performance can be compared across code versions as well as different hardware/software configurations.

3 Existing technology

A number of tools already exist that can do part of what is needed for this framework. These fall under 4 main categories: automation, performance analysis, data aggregation and visualisation.

3.1 Automation

3.1.1 GitHub Actions and self-hosted runners

For codes hosted on GitHub, a service called GitHub Actions[2] allows scripts to be run automatically when triggered by predefined events, for example when a new commit is pushed to a particular branch of a repository. By default these run within containers on GitHub-provided cloud infrastructure, but they can also be configured to run on external machines, known as self-hosted runners[3], provided the user has sufficient access to that machine. This allows users to have these Actions run on, for example, personal workstations or HPC facilities.

3.1.2 ReFrame HPC

In order to build and run the proxy apps on a particular system, the runner must have knowledge of how the system is configured. For example, which particular versions of compilers and libraries are available and where they are stored. While for single user machines these often live in default locations, more complex systems such as most HPC facilities will use system such as Environment Modules[4] or LMod[5] to manage multiple software environments. As well as this, such HPC facilities often provide access to computational resources via a queuing system, such as Slurm[6]. As these configurations are usually site-specific, any automated runner must be made aware of how to compile and run applications on a per-site basis.

ReFrame[7] is a regression testing and benchmarking framework which allows site configuration and application run-time configurations to be defined independently of each other, while also including integration of many common tools for managing software environments, build systems and queuing systems.

3.2 Performance analysis

There are many different types of performance measures that can relevant when developing software, for example application run-time, memory usage, parallel efficiency and more. As such, a wide range of utilities are available to analyse programs to quantify these. These range from simple system tools such as the Unix "time" utility that simply measures the amount of time a program takes to run, up to tools such as Intel VTune[8] that can perform much deeper analysis of a program's run-time performance. Often, utilities like this are available on HPC facilities.

3.3 Data aggregation and visualisation

In order to compare performance metrics from multiple applications across multiple sites, the data needs to be gathered and displayed in a web dashboard. In order to gather the data, it needs to be writable and accessible to the dashboard for visualisation. This can be done simply by hosting a static structured data file, for example on GitHub, or using more advanced databases such as MySQL or PostGRESQL services.

Displaying the data in an easy-to-understand way can be done in a number of ways as well. These range from a simple static web page using JavaScript graphing and visualisation packages such as Chart.js[9] or D3.js[10], up to full dashboard solutions such as Grafana[11], which will need more advanced hosting solutions.

4 Design & Implementation

4.1 Overview

In order to make use of existing technologies and make it as easy as possible for application developers to use these tools, a framework called RoundTable was designed to allow existing tools to be used together, with simple per-application configuration files describing the details of each application.

The RoundTable framework makes use of self-hosted GitHub runners on HPC facilities to run a range of benchmarks on applications, setting up Re-

Frame benchmarks for a range of performance analyses, parsing the results and gathering them to be displayed in a web dashboard.

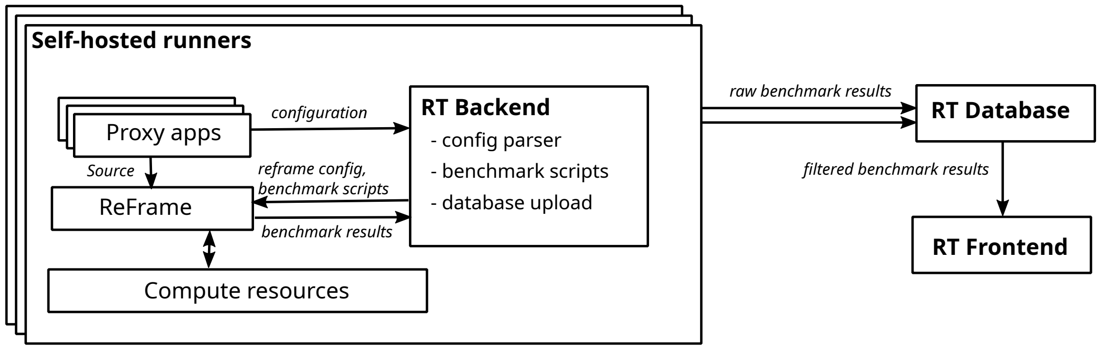

Figure 1: Diagram showing how all the components of RoundTable work together, and the data that is passed between them

4.2 RoundTable backend

The RoundTable backend performs 3 main roles:

- Generating the ReFrame scripts based off each proxy app's configuration,
- Providing benchmark scripts to run various forms of analysis and parse the results,
- Upload the results to the RoundTable database.

4.2.1 Proxy app configuration

Proxy apps are configured using TOML configuration files, describing how to build and run the application, as well as the environments and benchmarks to run. An example is listed below:

```
# roundtable.conf
app = "simple_md"
build = "make"
environments = ["gnu", "intel"]
execution = "simple_md -i <INPUT>"
[benchmarks]
    [benchmarks.small]
        INPUT = "examples/small.in"
        analysis = ["vtune"]
        resuorces = [
            num_procs = 1,
            time = "0:1:0"
        ]

    [benchmarks.large]
        INPUT = "examples/large.in"
        analysis = [
            "runtime",
            "memory",
            ""
        ]
        resuorces = [
            num_procs = 10
            time = "0:8:0"
        ]
```

This defines an app called ”simple_md” which can be compiled with Make. It has two associated benchmarks:

- A small serial benchmark which is run through VTune, and
- A larger parallel benchmark that only measures the runtime and memory usage

4.2.2 Benchmark scripts

In order to simplify the configuration of the applications, benchmarks are wrapped up into generic scripts that can be run with any application on any supported system, returning the results in a common JSON format. For example, a simple benchmark to display the program runtime is listed below:

```
#!/usr/bin/bash
# Returns the application runtime in seconds
time=$(\
/usr/bin/time -p "$@" 2>&1 \
  | awk '/real/{print $2}' \
)

echo {"runtime:" $time}
```

Defining benchmarks in separate scripts like this allows new benchmarks to be added centrally and immediately used for a number of different applications. It also allows more complex analysis to be performed, such as parallel scaling runs, averaging over multiple runs etc.

4.2.3 Database upload

Once all the benchmarks have been run, the RoundTable backend will push the results, along with metadata about when and where the runs took place, to a centralised database.

4.3 Database

The results from all application runs across all sites are stored in a centralised database that the dashboard can get data from. This is currently just a flat JSON file in a GitHub repository, but can easily be extended to use more advanced databases such as MySQL or PostgreSQL.

4.4 RoundTable dashboard

The dashboard is a static web application written in React.js which gets data from the database and renders it as a series of graphs drawn using the Chart.js package. Since it is a static web app, it can easily be hosted on GitHub Pages. Below is a screenshot of the database, showing some dummy data.

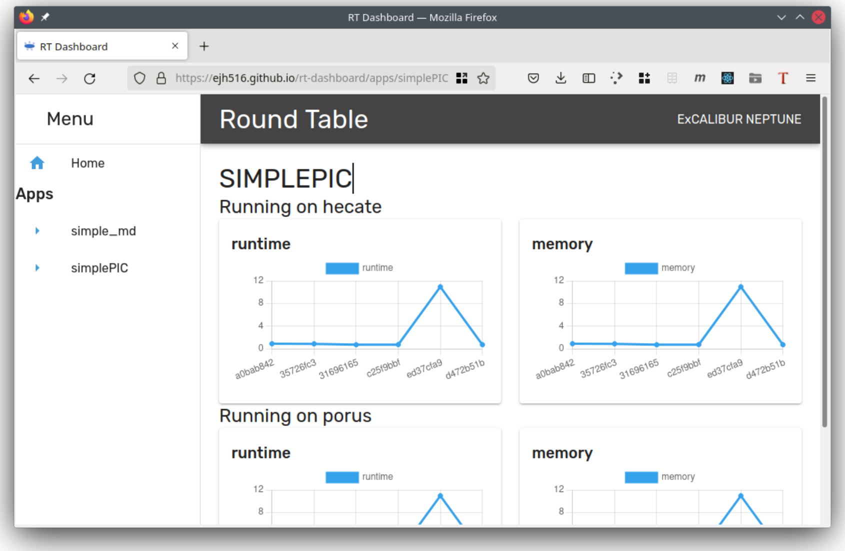

Figure 2: A prototype RoundTable dashboard showing dummy data for two applications run across two sites (hecate and porus).

4.5 Availability

The RoundTable framework will be available on GitHub soon, within the ExCALIBUR-NEPTUNE organisation.

5 References

1. *Excalibur Neptune website* https://excalibur.ac.uk/projects/excalibur-fusion-use-case-project-neptune-neutrals-plasma-turbulence-numerics-for-the-exascale/
2. *GitHub Actions* https://docs.github.com/en/actions
3. *GitHub Actions self-hosted runners* https://docs.github.com/en/actions/hosting-your-own-runners/about-self-hosted-runners
4. *Environment modules* https://modules.readthedocs.io
5. *LMod* https://lmod.readthedocs.io
6. *The SLURM scheduler* https://slurm.schedmd.com/documentation.html
7. *ReFrame HPC* https://reframe-hpc.readthedocs.io
8. *Intel VTune* https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html
9. *Chart.js graphing library* https://www.chartjs.org/
10. *D3.js visualisation library* https://D3.js
11. *Grafana dashboards* https://grafana.com/