The NOνA software testing framework

M Tamsett[1], **C Group**[2]
[1]University of Sussex, Falmer, East Sussex, UK
[2]University of Virginia, Charlottesville VA, USA

E-mail: `m.tamsett@sussex.ac.uk`

Abstract. The NOνA experiment at Fermilab is a long-baseline neutrino experiment designed to study ν_e appearance in a ν_μ beam. NOνA has already produced more than one million Monte Carlo and detector generated files amounting to more than 1 PB in size. This data is divided between a number of parallel streams such as far and near detector beam spills, cosmic ray backgrounds, a number of data-driven triggers and over 20 different Monte Carlo configurations. Each of these data streams must be processed through the appropriate steps of the rapidly evolving, multi-tiered, interdependent NOνA software framework. In total there are greater than 12 individual software tiers, each of which performs a different function and can be configured differently depending on the input stream. In order to regularly test and validate that all of these software stages are working correctly NOνA has designed a powerful, modular testing framework that enables detailed validation and benchmarking to be performed in a fast, efficient and accessible way with minimal expert knowledge. The core of this system is a novel series of python modules which wrap, monitor and handle the underlying C++ software framework and then report the results to a slick front-end web-based interface. This interface utilises modern, cross-platform, visualisation libraries to render the test results in a meaningful way. They are fast and flexible, allowing for the easy addition of new tests and datasets. In total upwards of 14 individual streams are regularly tested amounting to over 70 individual software processes, producing over 25 GB of output files. The rigour enforced through this flexible testing framework enables NOνA to rapidly verify configurations, results and software and thus ensure that data is available for physics analysis in a timely and robust manner.

1. Introduction

The NOvA experiment is a currently active long-baseline neutrino oscillation experiment using the recently upgraded NuMI beam at Fermilab, USA to measure $\nu_\mu \to \nu_e$, $\overline{\nu}_\mu \to \overline{\nu}_e$, $\nu_\mu \to \nu_\mu$ and $\overline{\nu}_\mu \to \overline{\nu}_\mu$ oscillations [1]. The experiment uses two functionally identical detectors composed of alternating horizontal and vertical planes of PVC plastic cells. Each cell has a 4×6 cm cross section and is filled with liquid scintillator and a looped wavelength-shifting fiber attached to an avalanche photodiode for light collection. The detectors are separated by 809 kilometers and located 14 milliradians off-axis from the beam center in order to produce a narrow-band 2 GeV beam near the oscillation maximum for ν_e appearance. The Far Detector located in Ash River Minnesota is 15.6m $\times$ 15.6m $\times$ 60m, totaling 14 kilotons and 344,064 individual cells. The Near Detector is located one km from the target at Fermilab and is 4.2m $\times$ 4.2m $\times$ 15.8m for a total of 300 tons and 20,192 cells.

2. NOνA software framework

The NOνA Offline Analysis Software (NOνASoft) is written in C++ and built on the Fermilab Computing Division's ART framework [2] that uses CERN's ROOT [3] analysis software. NOνASoft makes use of more than 50 external software packages, is developed by more than 50 developers and is used by more than 200 scientists from 38 institutions and 7 countries across 3 continents.

NOνA has already produced more than 1 million Monte Carlo and detector generated files amounting to more than 1 PB in size. These are divided between a number of parallel data streams such as far and near detector beam spills, cosmic ray backgrounds, several data-driven triggers and over 20 different Monte Carlo configurations. Each of these data streams must be processed through the appropriate steps of the rapidly evolving, multi-tiered, interdependent NOvA software framework. In total there are greater than 12 individual software tiers, each of which performs a different function and can be configured differently depending on the input stream.

3. Testing framework

In order to regularly test and validate that all of these software stages are working correctly NOvA has designed a powerful, modular testing framework that enables detailed validation and benchmarking to be performed in a fast, efficient and accessible way with minimal expert knowledge.

The core of this system is a novel series of python modules which wrap, monitor and handle the underlying C++ software framework. These are configured using command line options to select from a number of pre-configured chains which define the input data stream to use. Each chain is itself composed of a number of tiers which each consist of an individual call to the underlying NOνASoft framework. Tiers typically consist of a discrete stage of event processing, such as Monte Carlo generation, translation from the data acquisition to the offline data format, the reconstruction of physics objects or particle identification. Tier-to-tier differences are configured based on a small number of options including the configuration file to use, the preceding tiers upon which this tier is dependent, any commands that need to be executed in advance of the tier and the output files that this tier will produce. The configuration of chains is simple and flexible and is trivially extensible to easily cater for new data streams and tiers.

The desired tiers are run sequentially with each NOνASoft executable call being spawned using the python sub-process module. The child process is then tracked via its process identifier. The memory, disk and CPU usage are monitored while at the same time the output and error message streams are captured and injected with timestamps. These timestamps are later used to correlate the tracked metrics with key stages in the NOνASoft framework process, such as database calls, object initiation and the looping over events.

Once a tier has concluded, successfully or otherwise, its return code is logged and any output files it produced are analysed in terms of their size and contents. This enables NOνA to check that the desired objects are being created and also to monitor the disk space requirements of these. Furthermore checking of the sanity of output files allows downstream tiers to respond accordingly and not be falsely blamed for upstream failures. Once every tier in a chain has concluded all of the resultant metrics, output files and logs are archived ready for later analysis.

The testing framework is executed following nightly builds of the NOνASoft development branch and also following the building of any release version. The suite of tests executed is chosen to exercise all of the core data streams and software tiers and is run on the same batch system as production and user analysis jobs. In total upwards of 14 individual streams are regularly tested amounting to over 70 individual software processes. These produce over 25 GB of output files each day.

4. Visualisation of results

Tests executed nightly and on stable software release versions are reported via a web-based interface. This interface utilises modern, cross-platform, visualisation libraries to render the test results in a meaningful way.

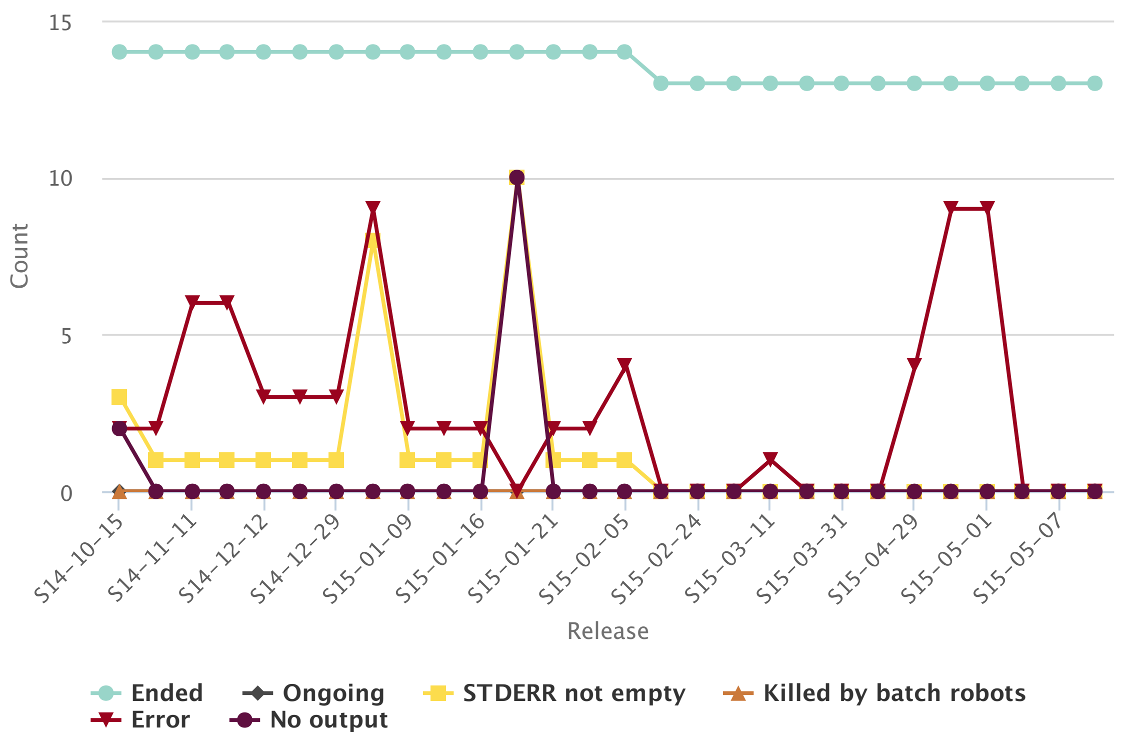
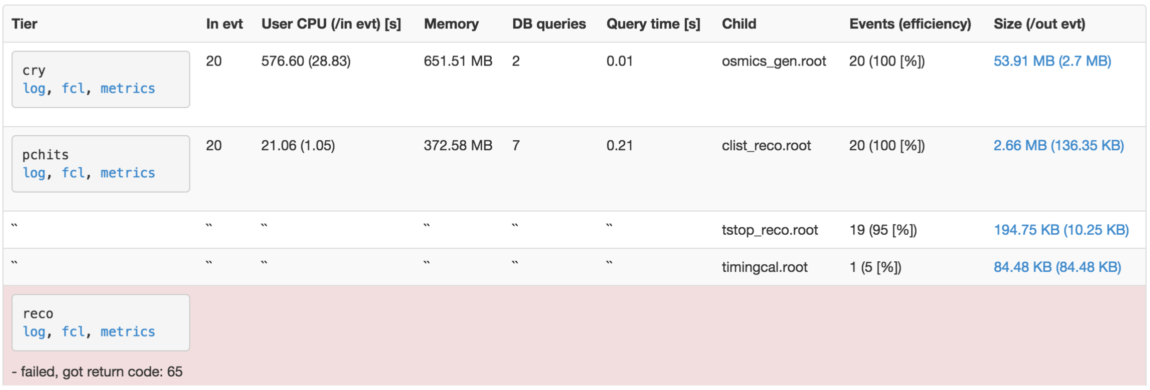


Figure 1. An example of a test summary graphic showing the collected results of many tests on static snapshots of the software framework over five months. This graph is made using the Highcharts JavaScript library [4].

Cron jobs periodically check on the outputs of all tests and use custom python modules to render the test results into HTML and JavaScript web pages formated using the BOOTSTRAP [5] development tools. These pages are self documenting, providing the user with a detailed break down of the configuration of all tests run.

The front page of the web-interface presents two overview graphics, the first showing a high-level summary of all tests performed on all NOνASoft release versions and the second similarly showing the results of the previous two weeks worth of nightly tests. An example is show in figure 1. Provided alongside this are links to the detailed results page of each suite of tests. Each results page begins with a mid-level summary of the results of all chains, along with links to the details of the batch jobs used to run these tests and their associated log files. Further down the page are low-level breakdowns of the test results of each of the tiers that form each chain. These provide the configuration used to run each tier, along with the full standard output and error streams produced during their running and the returned error code for failed tests. Also presented are the peak memory usage of each tier, the integrated and per event CPU times, the efficiency of the tier in terms of the ratio of input to output events and the number and length of database calls executed by this tier. Figure 2 shows an example of the summary table of a chain.



Tier	In evt	User CPU (/in evt) [s]	Memory	DB queries	Query time [s]	Child	Events (efficiency)	Size (/out evt)
cry log, fcl, metrics	20	576.60 (28.83)	651.51 MB	2	0.01	osmics_gen.root	20 (100 [%])	53.91 MB (2.7 MB)
pchits log, fcl, metrics	20	21.06 (1.05)	372.58 MB	7	0.21	clist_reco.root	20 (100 [%])	2.66 MB (136.35 KB)
"	"	"	"	"	"	tstop_reco.root	19 (95 [%])	194.75 KB (10.25 KB)
"	"	"	"	"	"	timingcal.root	1 (5 [%])	84.48 KB (84.48 KB)
reco log, fcl, metrics - failed, got return code: 65								

Figure 2. A summary table showing the results of a single test. In this case the chain tested was the Monte Carlo simulation of a far detector cosmic ray events. The chain was composed of three tiers. The first, *cry* [6] is a cosmic ray Monte Carlo generator. The second *pchits* is the limited reconstruction tier used in the calibration of the NOνA detectors. The third *reco* is the detailed reconstruction of physics objects for use in oscillation background studies. In this case the first two tiers passed the tests while the third failed. Details on the cause of the failure are available to the user via the provided links.

Further details on the performance of each tier is provided in a linked page which displays metrics such as CPU and memory usage as a function of time. An example of such a metric is shown in figure 3. These interactive graphs are given context via the analysis of the timestamped standard output message streams of each tier. Important events identified within the processing are represented as coloured circles displayed on the metric charts at the corresponding time. These provide the user with snippets of the message stream on mouse over.

Information on output files is provided in the summary table in terms of their total and per event size. A further page visualises this in a lot more detail by providing an interactive, zoomable starburst and treemap graphic. An example of this is shown in figure 4.

The benchmarking information reported by the tests is further utilised to forecast the future computational needs of the experiment in terms of integrated CPU usage, memory footprint and storage volume needed, which information allows NOvA to rapidly and accurately predict their future needs and to plan production campaigns accordingly.

5. Conclusions

This paper has presented the NOνA software testing framework. The rigour enforced through this flexible testing framework enables NOvA to rapidly verify configurations, results and software and thus ensure that data is available for physics analysis in a timely and robust manner.

Acknowledgements

The author acknowledges support for this research was carried out by the Fermilab scientific and technical staff. Fermilab is Operated by Fermi Research Alliance, LLC under Contract No. De-AC02-07CH11359 with the United States Department of Energy

References

[1] D. S. Ayres *et al.* [NOvA Collaboration], FERMILAB-DESIGN-2007-01.
[2] C. Green and J. Kowalkowski and M. Paterno and M. Fischler and L. Garren and others, J. Phys. Conf. Ser.,396,022020 (2012).

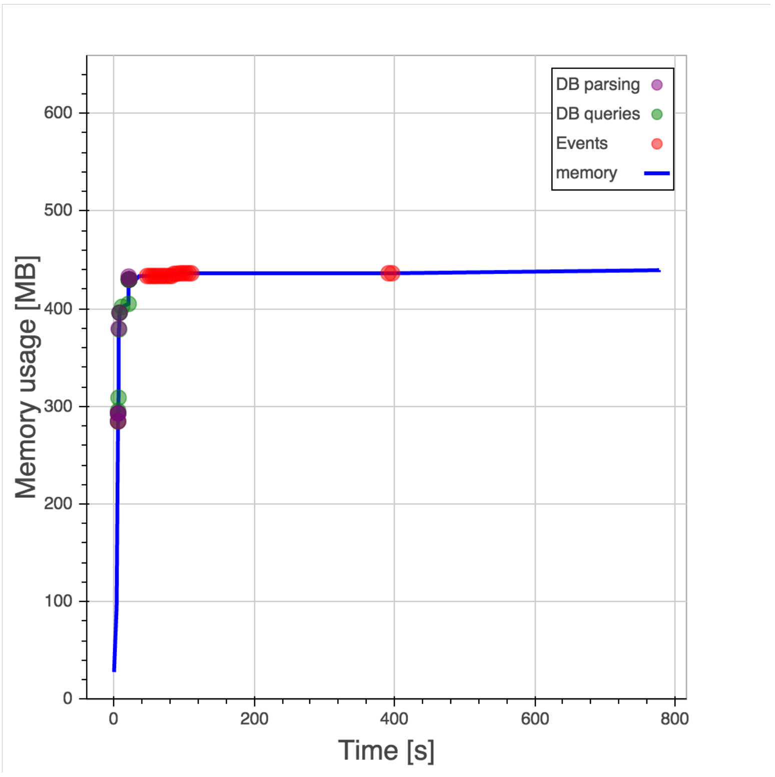


Figure 3. The memory usage of one test visualised using the Bokeh python library [7]. The coloured circles are displayed at times when important processing steps occur. These provide the user with contextual information on mouse hover.

[3] R. Brun and F. Rademakers, Nucl. Instrum. Meth. A **389**, 81 (1997).
[4] Highcharts, http://www.highcharts.com/.
[5] Bootstrap, http://getbootstrap.com/.
[6] Hagmann C, Lange D, Wright D Cosmic-ray Shower Library (CRY), LLNL UCRL-TM-229453 Lawrence Livermore National Laboratory. Avaliable at http://nuclear.llnl.gov/simulation/cry.pdf
[7] Bokeh, http://bokeh.pydata.org/en/latest/index.html.
[8] Data Driven Documents, http://d3js.org/.

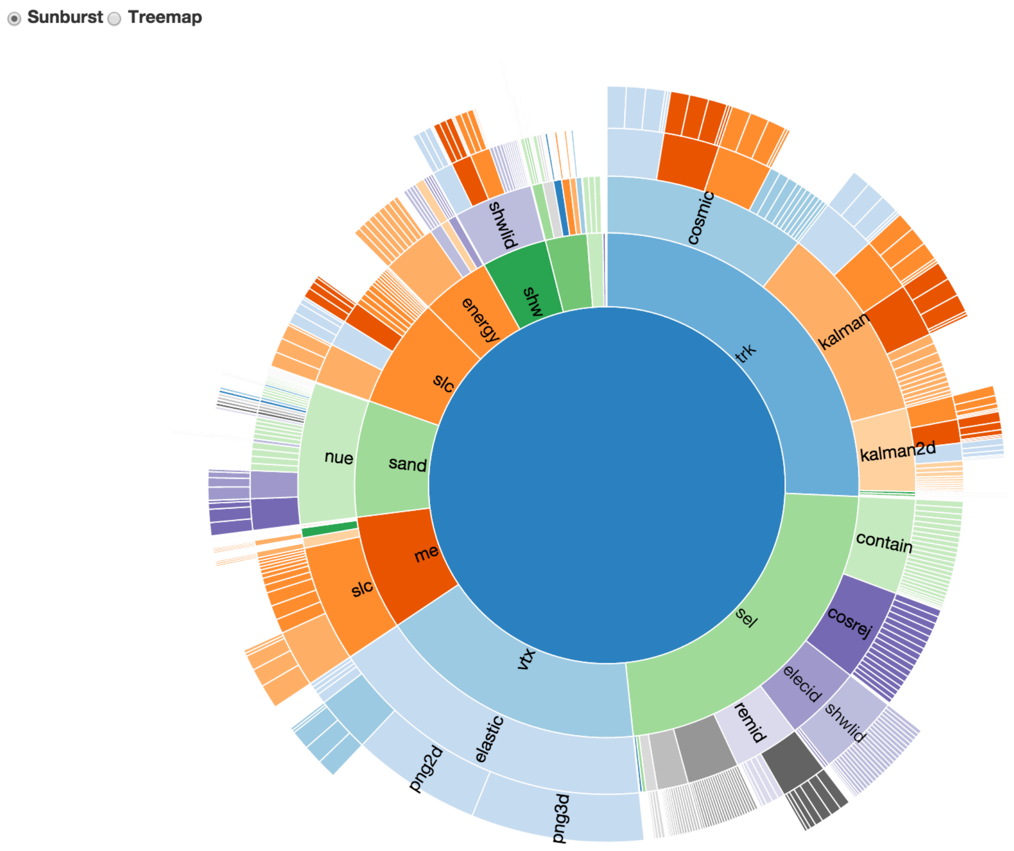


Figure 4. A D3 [8] based graphic displaying the size of one data format produced during tests. This graphic allows users to drill down to study the particular portion of the data format they are concerned with.