Continuous Performance Benchmarking Framework for ROOT

Oksana Shadura, Vassil Vassilev, Brian Paul Bockelman
oksana.shadura@cern.ch, vvasilev@cern.ch, bbockelm@cse.unl.edu

CHEP 2018, Sofia, Bulgaria

Motivation

Foundational software libraries such as ROOT are under intense pressure to avoid software regression, including performance regressions. *Continuous performance benchmarking, as a part of continuous integration and other code quality testing, is an industry best-practice* to understand how the performance of a software product evolves over time. We present a framework, built from industry best practices and tools, to help to understand ROOT code performance and monitor the efficiency of the code for a several processor architectures. It additionally allows historical performance measurements for ROOT I/O, vectorization and parallelization sub-systems.

Project Goals

We aim to provide:

1. Continuous performance monitoring of ROOT components;
2. Speculative performance monitoring on opened pull requests (PR);
3. Rich and customizable visualizations to aid performance analysis.

Implementation

We utilize the Google benchmarking library[1]to execute micro benchmarks of selected hotspot functions in ROOT and related libraries. This provides detailed data measurements, including memory usage and CPU instruction counters. Additionally, the framework manages traditional benchmarking pitfalls via repeating unstable benchmarks and providing a stable performance environment over time. The performance data points from continuous benchmarking are fed into an InfluxDB database and provided to the developer community via a Grafana-based dashboard. This performance benchmarking framework, built on generic and flexible infrastructure, is meant to be reusable by other projects. The tool can be connected to Jenkins or other similar services as shown in Fig 1.

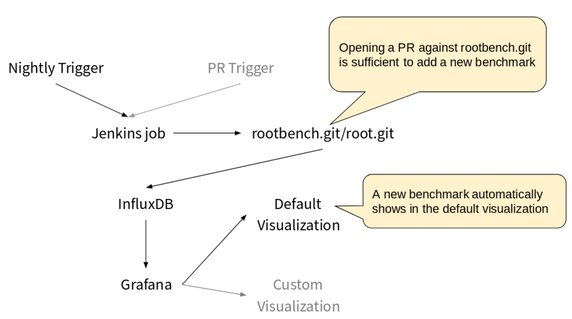

Figure 1: Jenkins – ROOTBench Flow.

Results

We have defined a basic set of measurements: Real Time (RT); CPU Time and RSS memory footprint. The set of performance indicators could be expanded with custom ones. Google Benchmark enables multi-threading analyses and addition of custom parameters such as number of branches in a generated TTree.

ROOTBench: Continuous perf. monitoring of ROOT

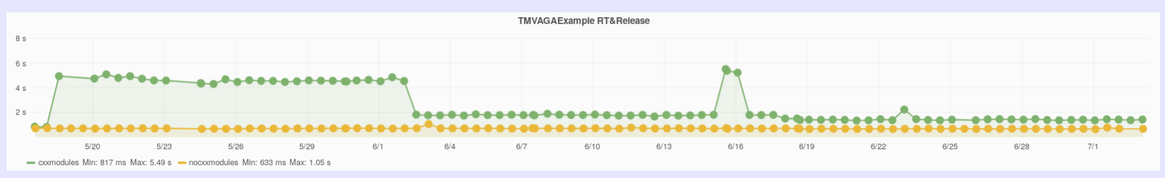

Figure 2: Performance improvements (enhancements in runtime) in Interpreter benchmarks

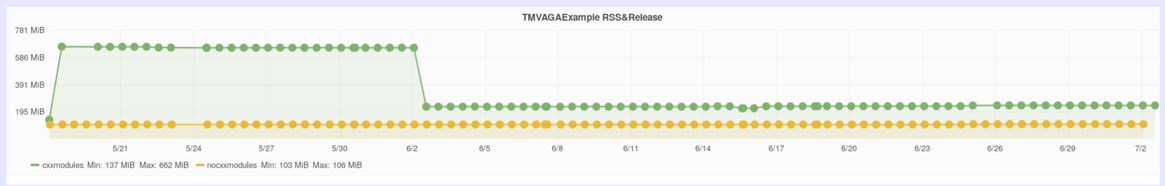

Figure 3: Memory Footprint Comparison Between ROOT runtime_cxxmodules and pch Features

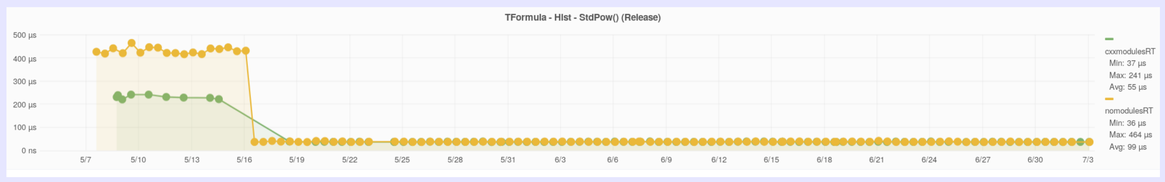

Figure 4: TFormula Speedup.

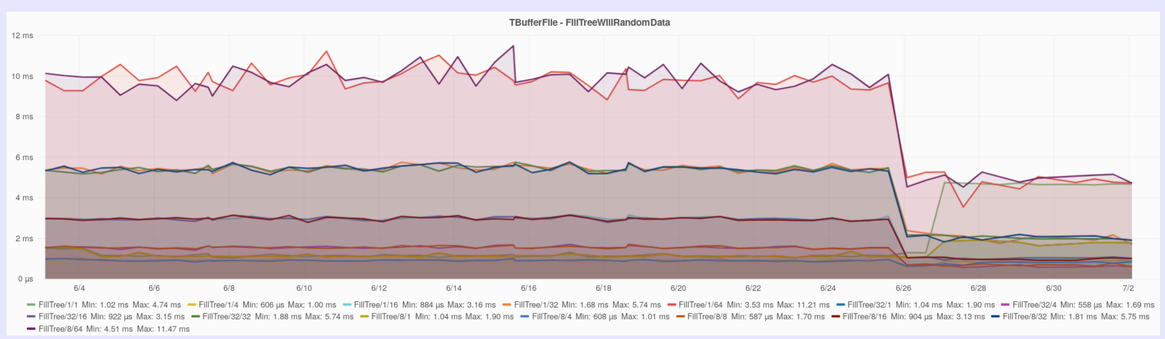

Figure 5: Threading Improvements in TBufferMerger

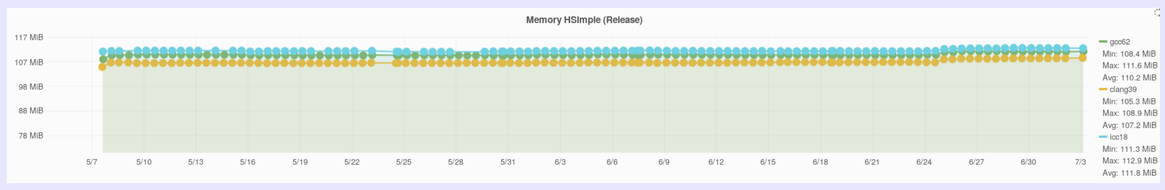

Figure 6: Memory Footprint of clang-, icc- and gcc-compiled ROOT Running HSimple

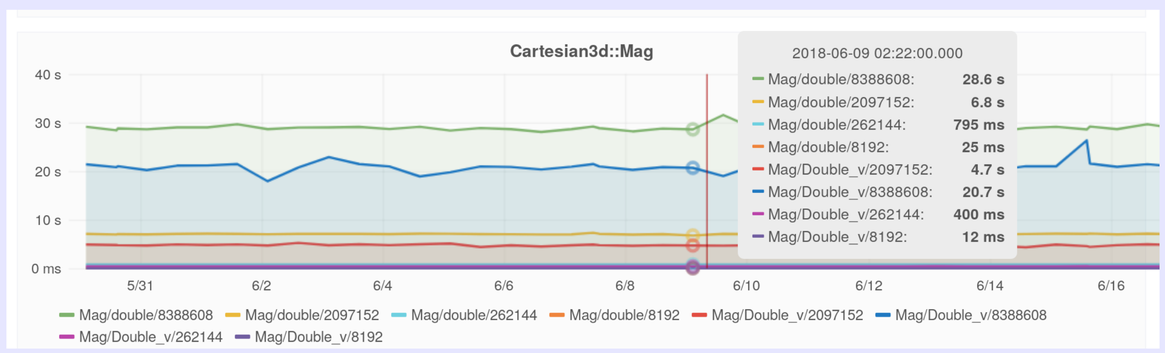

Figure 7: RT comparison of vectorized and scalar code in GenVector::Mag()

Fig 2-7 show several real-world examples:

- Interpreter – implementation of Bloom filter as per PR2093, PR2137 and PR2204 (Fig 3)
- TFormula – remove debug check in TFormula as per from PR2017 (Fig. 4)
- TBufferMerger – remove callback functionality to avoid over subscription of the machines as per PR2245 (Fig. 5)
- Hsimple benchmark – comparison of memory footprint of ROOT compiled with different compilers (Fig 6)
- GenVector – comparison of scalar and vectorized implementation in GenVector library (Fig. 7)

Conclusion

The prototype is very functional and it has been often used to find performance degradation. Performance sensitive code can be monitored at the cost of a standard pull requests against the ROOTbench GitHub repository. Several external benchmarking contributions have been submitted some of them track TMVA and RDF components. We have enumerated several important future improvements:

- Enable the current infrastructure to work on pull requests (PR);
- Introduce alerts if metrics have gone beyond defined thresholds;
- Complement performance graphs with performance flame graphs;
- Introduce more benchmarks statistics to stabilize the RT metrics;
- Introduce performance node monitoring to monitor regressions in the host hardware to reduce noise;
- Implement versioning of dashboards (dashboards auto generation via JS).

Performance of large-scale systems is fragile and can vary on the different systems. It is vital for the projects to offer a set of tools and benchmarks allowing coders to reason about performance. We hope ROOTBench is a step towards recognizing and solving the problem ensuring better sustainability of HEP Software.

Acknowledgement

The presented work was funded by IPCC-ROOT[2] and DIANA-HEP [3]. The authors would like to thank Openlab [4] for providing dedicated machines for continuous performance monitoring.

References

[1] Google benchmark. `https://github.com/google/benchmark.git`.

[2] Ipcc root. `https://ipcc-root.github.io/`.

[3] Diana-hep. `http://diana-hep.org/`.

[4] Openlab. `https://openlab.cern/`.

[5] Root benchmark project. `https://github.com/root-project/rootbench.git`.

[6] Root project. `https://github.com/root-project/root.git`.