Continuous performance monitoring

Vassil Vassilev

Motivation

- Enabling performance optimization contributions (often external) to ROOT
- Making sure these contributions are sustainable (i.e. once the money is spend for optimization it shouldn’t regress)
- Providing continuous performance monitoring service

Approach

- Use an external benchmarking library
- Register machines with exclusive access to jenkins. We have been granted exclusive access to an Intel Haswell and KNL machines in OpenLab.
- Implement micro and macro benchmarks.

Benchmark

- A program which tests code scalability
- Two major kinds: micro and macro benchmarks
  - Micro benchmarks are like unit tests (eg. googletest) but for making sure certain routines (or small set of routines) scale
  - Macro benchmarks are like integration tests (eg. roottest)

Benchmarking Libraries

- Unfortunately not a lot of specialized libraries on the market
- Google Benchmark stands out with a compatible, open-source license

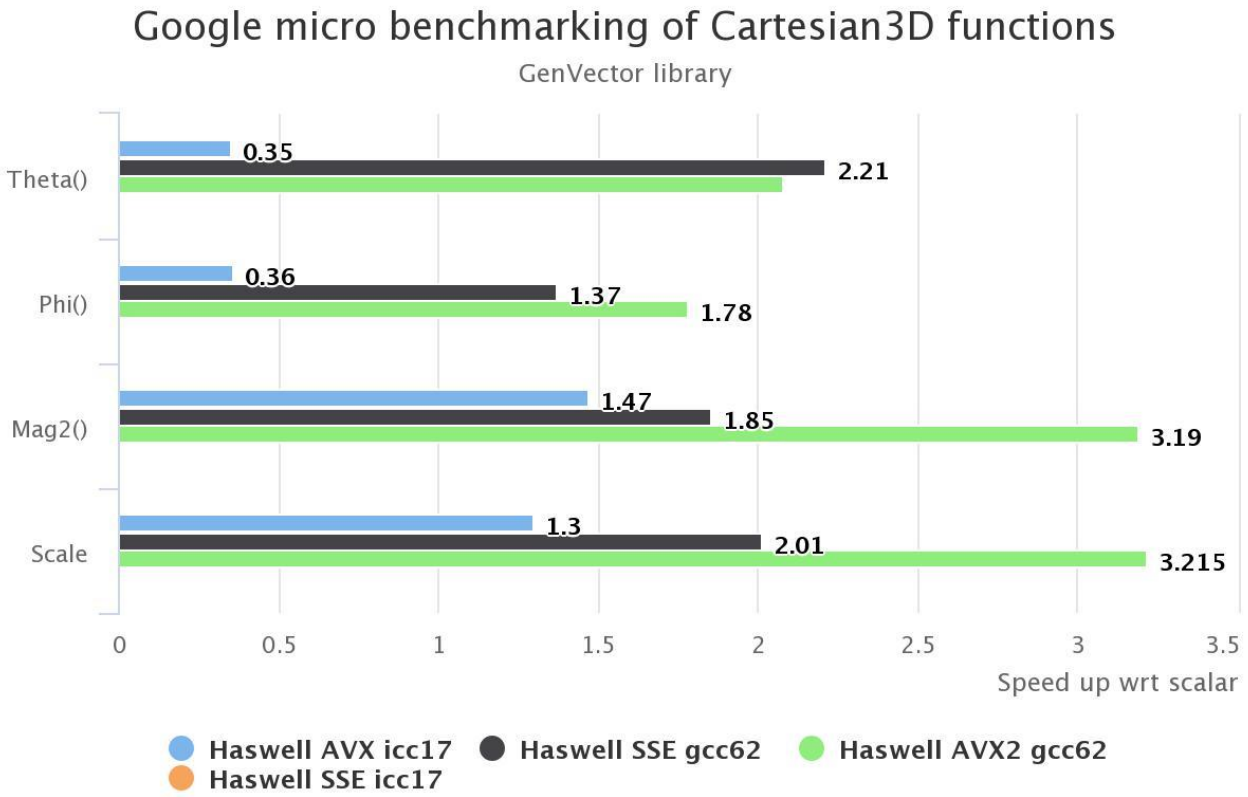
Macrobenchmarks

- They are complex programs which can possibly run long time
- They can have dependencies to external packages (eg. tensorflow, protobuf, etc) and compare performance
- They should live in rootbench.git in a similar to roottest manner.

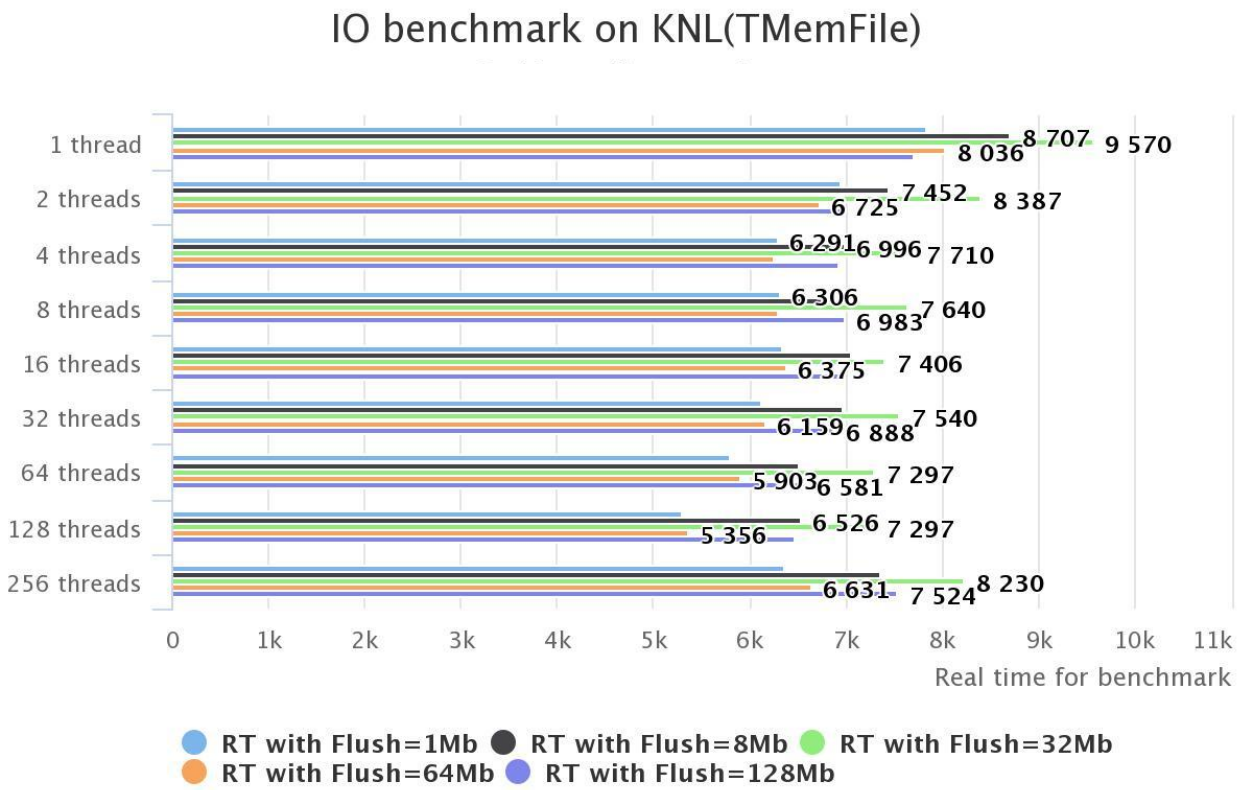
Microbenchmarks

- Small, easy-to-write programs with dependencies only in ROOT
- They can compare performance relative to a mode (eg. single thread vs multithreading, scalar vs vector code, etc)
- They should live in root.git in a similar to gtest manner

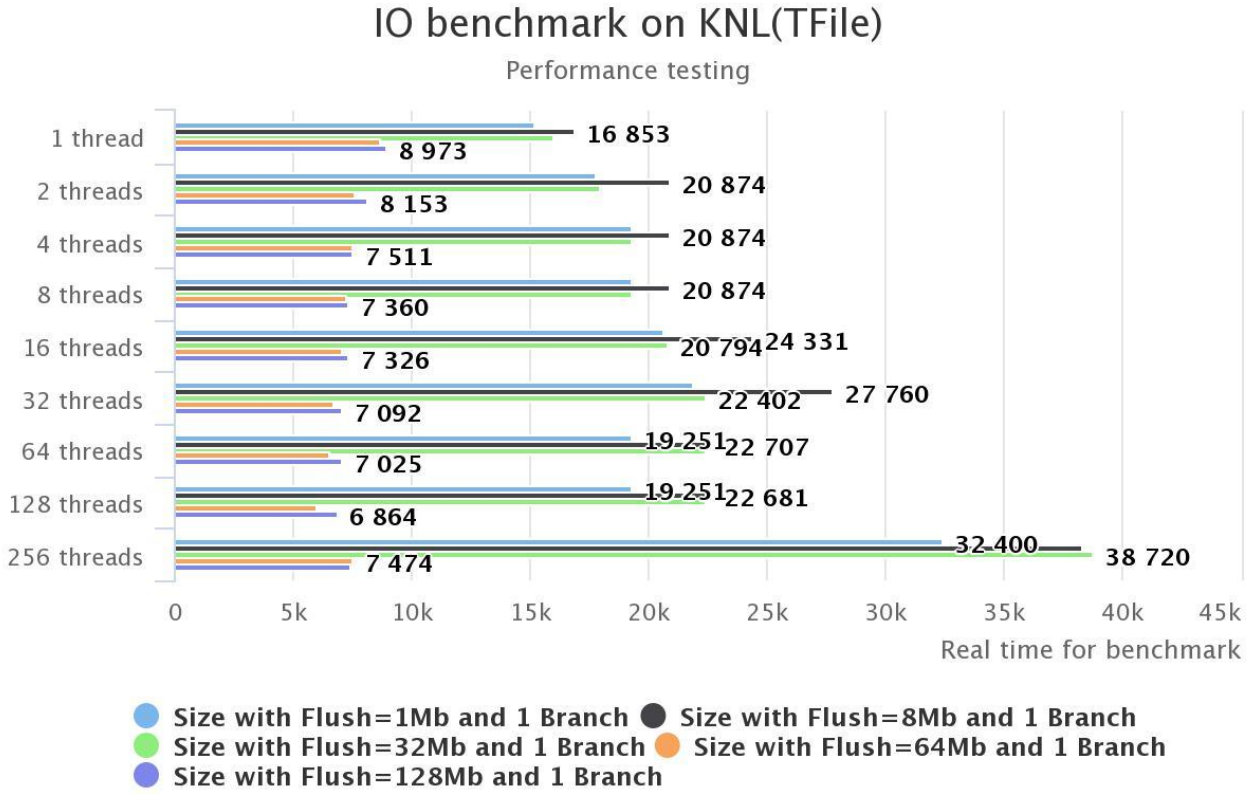
GenVector microbenchmarking



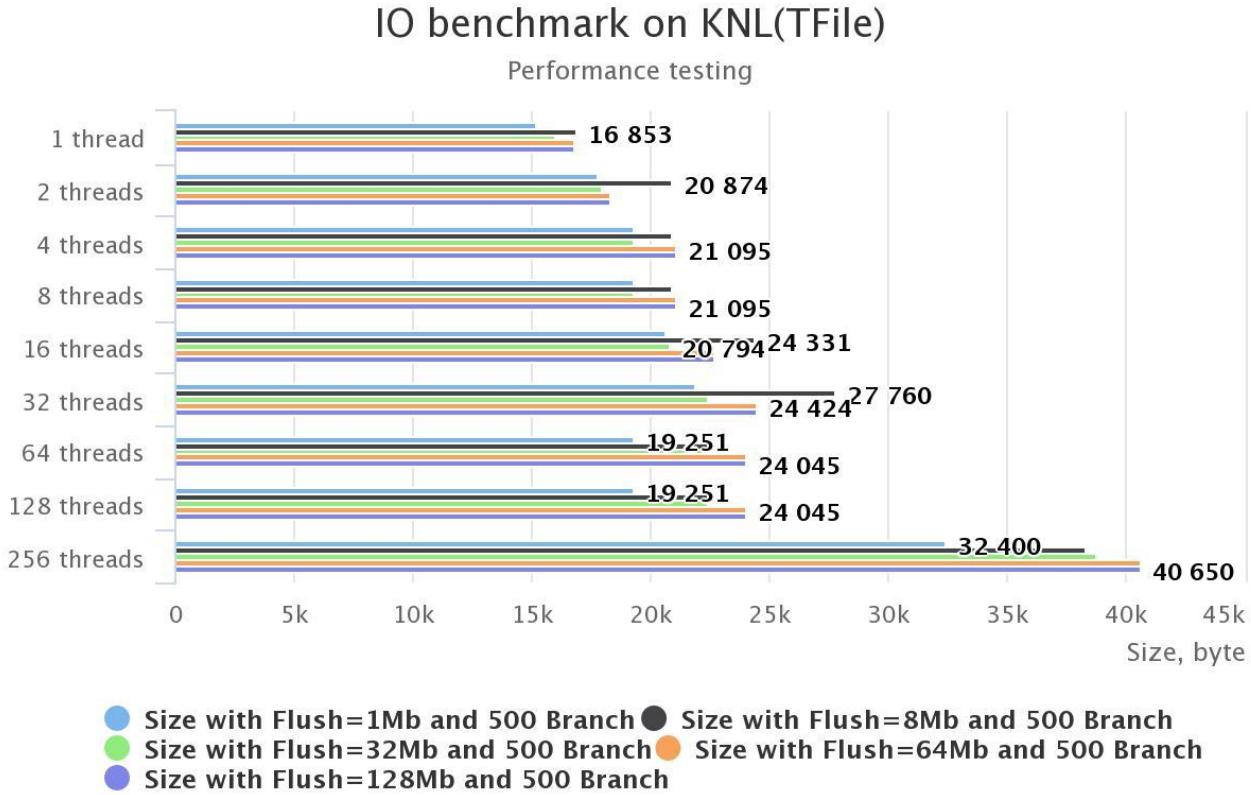
I/O microbenchmarking



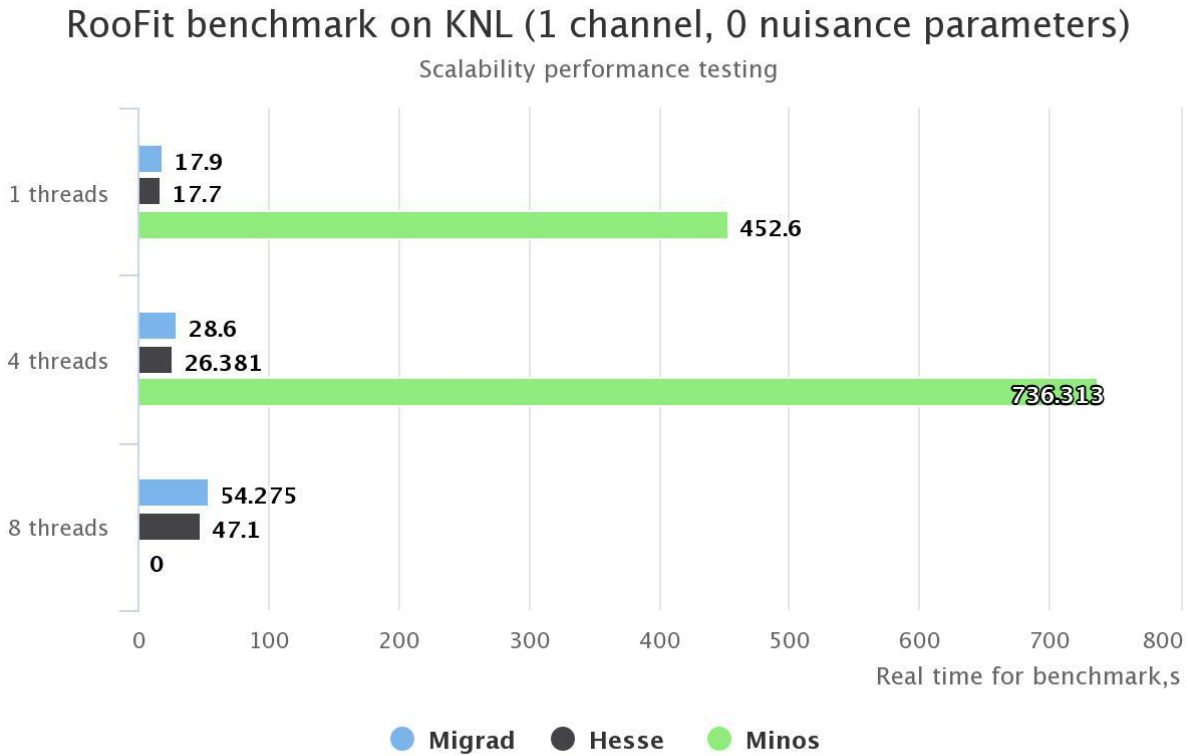
I/O microbenchmarking



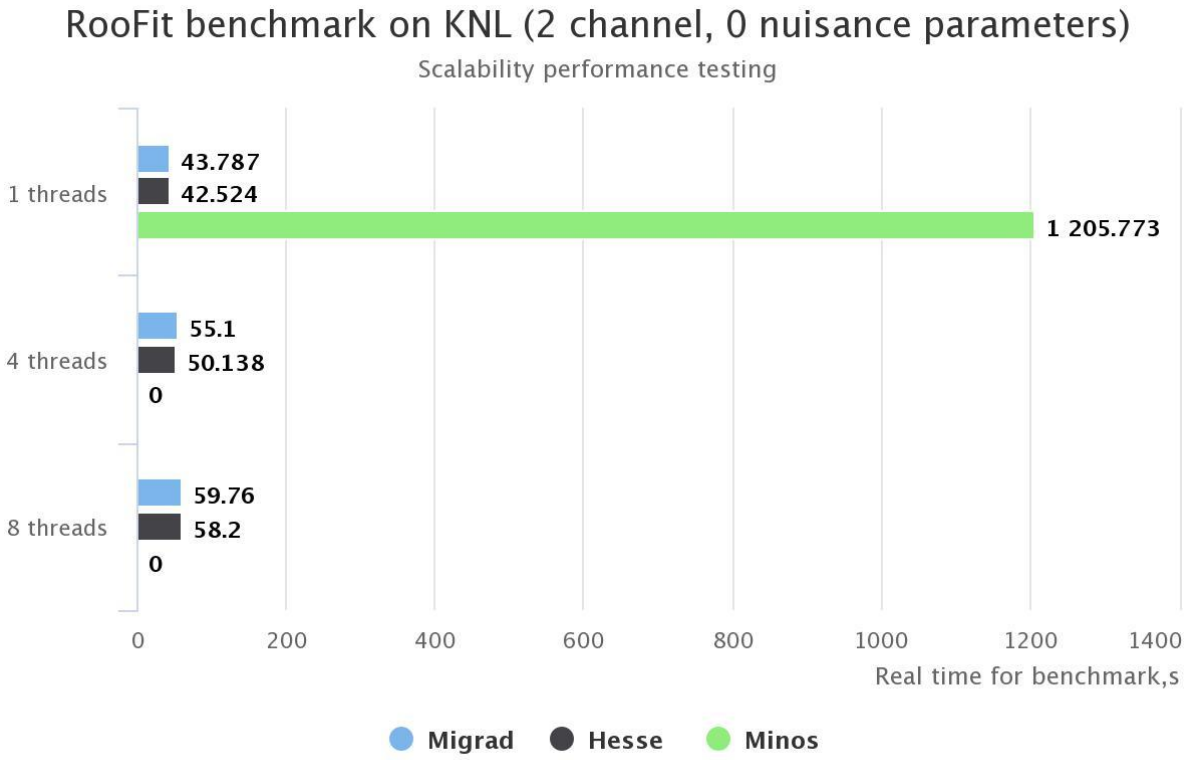
I/O microbenchmarking



RooFit benchmarking



RooFit benchmarking



Infrastructure for data analytics and visualization of benchmarks/tests

Oksana Shadura

Goals

- Save statistics of job/test/benchmark runs
- To be able provide visualization of results
- To be able to detect regression and calculate statistics
- To add a performance check to PR or build incrementally each number of commits

New Jenkins slaves for performance testing

- Thanks to Openlab, it was provided 2 powerful nodes:
  - Haswell (56 cores)
  - KNL (64 cores)

We will create special label for them!

New Jenkins job: how we can do it

- Big matrix of possible test cases (compilers/SIMD/other flags)
- Build as a part of PR or incrementally during day
- Google benchmark has a nice JSON/CSV output
- Possibility to trigger job via PR (other triggers) or just directly generate performance plots (data for performance plots)

Available resources https://monit.cern.ch/

Services

MONIT-KIBANA

Use Kibana to explore metrics stored in Elasticsearch and interactive data discovery and plotting.

MONIT-GRAFANA

Use Grafana for plotting time series metrics coming from different datasources like Elasticsearch or InfluxDB.

MONIT-ZEPPELIN

Use Zeppelin to generate more advanced reports and visualizations over HDFS with the help of Spark.

MONIT-TIMBER

Use Kibana to explore logs stored in Elasticsearch and interactive data discovery and plotting.**(IT only)**

Grafana + InfluxDb for continuous storage of records

- XML reports could be send to Influxdb Jenkins Plugin+JUnit
- Or send data directly to DB via curl (influx-db target (e.g: url,desc,user,pwd,db,rentention) on Jenkins configuration using groovy script)

jenkins_data	• Build health • Build status message • Build time • Job duration • Build result • Build result ordinal (0=Stable, 1=Unstable, 2=Failure, 3=Not built, 4=Aborted) • Successful build boolean • Last stable build number (or 0 if never) • Last successful build number (or 0 if never) • Tests failed (unit test results from JUnit Plugin) • Tests skipped (unit test results from JUnit Plugin) • Tests total (unit test results from JUnit Plugin)	JUnit Plugin

No plans to invest a lot of time, but just **to be able to save data generated from builds** to database and after be able to visualize it on demand!

Generate plots on local machines (standalone)

- JSON to JSON Highchart friendly version (written by Rafael): **json-bench-converter**
- Easy Highchart generation on https://jsfiddle.net using pregenerated JSON

Thank you!

Backup slides

Pros and cons in having in-tree microbenchmarks

- This would allow us to use various thread sanitizers and analyses when compiling root. This is especially useful for header-only (or template heavy) code, such as TDF.
- Gives a handle to solve “*And btw I forgot one item in the failure causes you'll see: the benchmark itself is changed, or the code is expected to be slower because of say an additional runtime check. Happens all the time to *something* in ROOT, too, and any of those will mark Jenkins red, unless the tool somehow distinguishes this? For the old tooling that was a killer.*”
- Gives a relationship between PR and benchmarks
- Give us a chance to revive and improve the existing in-tree performance tests

Pros and cons in having in-tree microbenchmarks

- Easier to relate ROOT build options (eg. -O2, IMT enabled) to the benchmark executables
- Has greater contribution visibility, shows that ROOT considers performance a first-class citizen
- Makes creation of a test/benchmark one degree easier (no need to clone separate repo, set it up, etc)

Informal team survey

- Based on two polls I did over the last couple of months with various team members we have: 1 strongly against, 1 strongly in favor and few in favor and the majority neutral.