Systematic profiling to monitor and specify the software refactoring process of the LHCb experiment

Ben Couturier
CERN, CH-1211 Geneva 23, Switzerland
E-mail: `ben.couturier@cern.ch`

Emmanouil Kiagias
CERN, CH-1211 Geneva 23, Switzerland
E-mail: `emmanouil.kiagias@cern.ch`

Stefan B. Lohn
CERN, CH-1211 Geneva 23, Switzerland
E-mail: `stefan.lohn@cern.ch`

Abstract. The LHCb collaboration develops and maintains large software frameworks, critical to the good performance of the experiments, that face challenges due to the increase of throughput requested from the Physics side that will not be matched by the computing resources, and by new computing architectures such as many-core, that cannot be currently fully used due to the limited amount of memory available per core. In the coming years, a considerable refactoring effort will therfore be needed to vectorize and parallelize the code, to minimize hotspots and to reduce the impact of bottlenecks. It is crucial to guide the refactoring with a profiling system that gives hints to parts for possible and necessary source-code reengineering and which kind of optimization could lead to final success. From detailed profiling few results are selected, summarized and available to be visualized by a web analysis frontend.
Software optimization is a sophisticated process where all parts, compiler, operating system, libraries and chosen hardware play a role in. Intended improvements can have different effects on different platforms. To obtain precise information of the general performance, to make profiles comparable, reproducible and to verify the progress of performance in the framework, it is important to produce profiles more systematically in terms of regular profiling based on representative use cases and to perform regression tests. Once a general execution, monitoring and analysis platform is available, software metrics can be derived from the collected profiling results to trace changes in performance back and to create summary reports on a regular basis with an alert system if modifications led to significant performance degradations.

1. Introduction
The LHCb collaboration develops and maintains large software frameworks, critical to the good performance of the experiment. While the LHCb Software performed satisfactorily during the LHC Run1, it now faces several challenges: the LHCb Upgrade will imply more Data processing with the same amount of computing resources, and the Computing hardware is evolving with the introduction of many-core architectures, that cannot be currently fully used due to the limited amount of memory available per core. For this reason, in the coming years, a considerable refactoring effort will therfore be needed to vectorize and parallelize the code, to minimize hotspots and to reduce the impact of bottlenecks. It is crucial to guide the refactoring with a system that gathers profiling information and summarizes the results in order to make the analysis as easy as possible.
During its development phase, the LHCb Software was constantly optimized; the profiling was however the responsibility of each developer, with no “official” profiling test suite defined and no record of the results. While this approach was effective in the framework development phase, there are no record of the evolution in sofwtare performance, nor of the current baseline. With a refactoring of the code under way, and some of the initial developers having gone, it is therefore now necessary for LHCb to put in place systematic profiling tests, in order to ensure that there is no performance degradation. These tests have to be as automated as possible, and the framework should allow to spot differences in the software performance between the new versions and the baseline.
This paper describes the framwork developed by LHCb to profile its application and to display the results in a user friendly manner. This paper is organized ...

2. LHCb computing
2.1. LHCb software
The LHCb experiment software is based on the Gaudi[1], a C++ framework using generic and object-oriented features of C++ for computing intensive tasks, and python for configuring and structuring modulesi (algorithms, tools...). It executes consecutive an abstract series of these algorithms to process data objects from the transient store on request. Gaudi is providing core services and tools for applications to hide complexity and make future development and changes more transparent for users. It is a large-scale framework and is additionally used by ATLAS, Glast, Harp and other experiments.
Applications build on top of Gaudi are Brunel, Moore, DaVinci, Gauss, Boole and others. Brunel is responsible for the offline reconstruction, Moore is the implementation of the High-Level Trigger (HLT) to decide weather event data will be stored or not, DaVinci ???, Gauss to simulate the particle transport and interaction through several detector modules, and Boole performs the digitization.

2.2. Computing environment
The LHCb computing environment persists in particular out of the resources accessed via the Worldwide LHC Computing Grid (WLCG), Cloud Infrastructure and the HLT farm located close to the experiment. Some 100k CPU’s are involved in data collection of the different detector subsystems, event filtering, offline reconstruction, stripping and simulation. 35 GB/s of recorded data have to be processed by 1500 computing nodes of the Event Filtering Farm (EFF) of the HLT to be reduced to 70 MB/s [2].

2.3. Integrated Profiling
In HEP computing it is a common method to measure performance via throughput (events per time unit). Thus the performance analysis is focused on the time linear and not the time constant part of processing. To achieve this instrumentation is an important advantage for

profiling source-code in large scale frameworks like the applications from the LHCb experiment. Multiple profiling measures have been implemented in the Gaudi framework using the Auditor-Service, which provides an interface for executing code between the event processing.
Timing information from the operating system's process information are collected using the TimingAuditor and are printing a summary of time spend in the applications algorithms. Likewise information can be collected using the MemoryAuditor or MemStatAuditor for changes in memory as soon they are observed. Recent work [3] conducted by Mazurov and Couturier has shown to improve precision in profiling the event-loop by implementing instrumentation for Intel's VTune^{TM}Amplifier which can be added using the IntelAuditor. Another strategy is to collect information from the performance monitoring unit (PMU) of modern CPU architectures to collect information about hardware related issues, such as cache-misses, branch-misprediction and stall cycles as done by Kruse and Kruzelecki [4] for the Gaudi framework.
Many of such kind of work has been performed to provide tools for developers to profile their code. Still, systematic usage or comparative profiling has been sparely observed.

2.4. Systematic Profiling

Three important aspects must be considered as crucial for systematic profiling. Profiles must become *comparable*, *reproducible* and *representative* to allow regression analysis. For this purpose profiling must be limited to small number of default cases and a fixed set of reference data, to ease the observation of relevant hotspots and to avoid a profile to vary because of differing types of physics events or other parameters. This way differences in the execution behavior between two software revisions can be examined and traced back on changes in related source-code. On the other side, changing events could later on be used to evaluate the needs of computing resource if other types of event data, for the upcoming data-taking periods or other projects, are expected.
Furthermore, profiles must be reproducible to be able to compare the test configuration of executed test jobs. This affects the job configuration, to log the software/platform information, as well as the run configuration provided by option files for Gaudi applications. Finally, gathered information should be precise and reliable making a regular execution of a series of test jobs necessary.
Hence the LHCb PR project has the following requirements.

(i) The expected huge amount of information must be centrally collected and easily become accessible. This can be achieved by using a web application as interface to supported brief- and detailed analysis of collected data.

(ii) Profiling is a changing subject with new interesting technologies. A solution must be flexible to include new profiling tools. Information must be collected by parsing generated reports and hooking them onto a central database.

(iii) To reduce work generic ways of navigation and visualization have to be investigated.

(iv) Regular execution would be labor intensive without an automated execution chain. Automated triggering, setup and data collection of the profiling procedure simplifies systematic profiling attempts.

To fulfill the objectives and requirements, the LHCb PR project contains three important technical aspects. First, the PRConfig package was created to store the run configuration in a version control system and to collect further job information and final profiling results into a central SQL database. Second, a system to configure, trigger and submit test jobs and finally a web application to quickly access, visualize and propagate results.

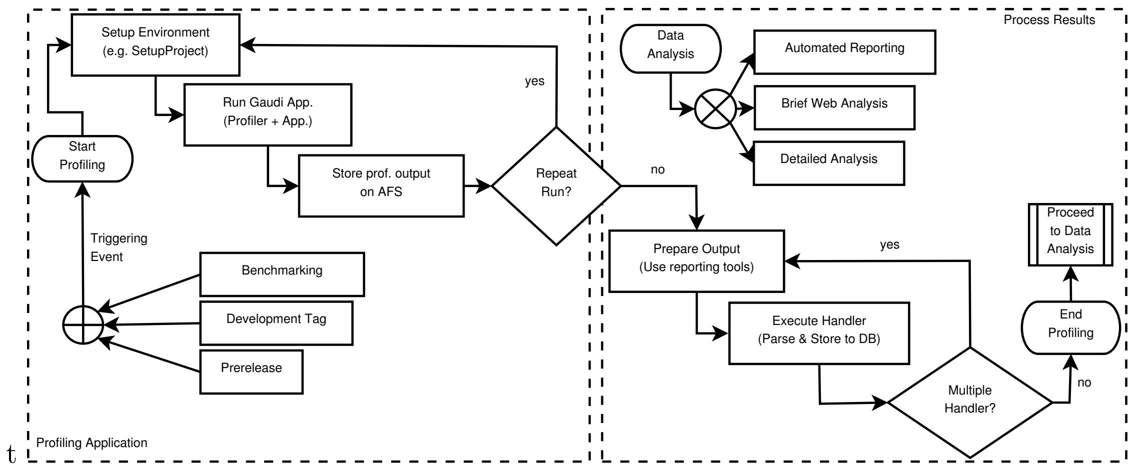

Figure 1. *nix*

3. Workflow and Implementation

The LHCb PR platform uses the continous integration system Jenkins [6] to prepare, configure and schedule jobs for profiling, a set of wrappers to customize available profilers for their working environment, a set of data handlers to parse and collect output data from the profilers, a SQL database to store summarized data and the web analysis framework LHCb PR.

- Job triggering and execution of profiling
- Definition of test or default cases
- A customize method for data collection
- Web-based analysis platform

3.1. LHCb PR framework

3.2. Job distribution and triggering

To facilitate regular and intensive systematic profiling, Jenkins is used to manage the job distribution to the test platform. This makes the in- or exclusion of other platforms simple and avoids interference between multiple runs on the same machine. The configuration (creation) of Jobs can further be used for a test specific pre-installation and compilation for development specific purposes or to run pre-configured jobs before new releases are tagged.
An other advantage is that the job configuration in Jenkins can be used for regular execution to call a validation test of recent builds from the build system to perform a subsequent profiling procedure.

3.3. Execution and Profiling

Commercial and open-source profilers becoming more and more available. The open source community developed crucial tools like the valgrind tool suit. Other tools like google's tcmalloc can be used to elaborate processing time and memory consumption. Additional, recent hardware features give access to hardware counters of the PMU (performance monitoring unit), which can be read from proprietary software like intels VTune or open source projects like oprofile.
This paper is not going to evaluate the benefits and drawbacks of different tools, but wants to enable a profiling platform like LHCb PR to individually setup these tools on their specific necessary way. For these purposes scripts are collected into a separate repository which can easily execute the test cases and stay flexible for individual configuration. To improve the configuration

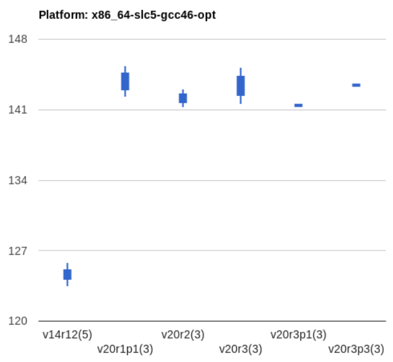

Figure 2. *nix*

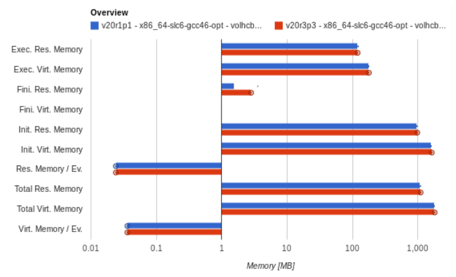

Figure 3. *nix*

of profiling runs and to focus the profiling onto main time consuming parts, integration of the profilers instrumentation methods are highly recommended.

3.4. Data collection
Data collection can be done in three different ways, first one can segregate information from the results of profilers, store files containing performance information and store the resting profiler specific collected information. Segregating information can also be quite diverse depending on which profiler were delivering the information. To maintain flexibility here, a collection of data handler were written for each profiler in use. They have to parse the output, select and combine information and finally collect it for insertion into the LHCb PR underlying database.

3.5. Test cases
Use cases are important to trace back performance changes to the evolving algorithms during the source code refactoring period. Test cases shell base on default use cases, which in deed, must not necessarily exist for frameworks. Still, all test cases should be best approximations to production usage. But sophisticated environments can influence runs on an non-deterministic way, which hampers the profile analysis. The highest priority of the current systematic profiling is to find software related issues, that can be addressed or have to be taken into account for upcoming decisions in resource allocation.

3.6. Use case Reconstruction
3.7. Use case High-Level-Trigger
Unfortunately the HLT framework Moore can not simply be reduced to a view common default cases, what makes it more difficult to trace back performance issues that way as it would affect production. The HLT brings further complications for the PR project in two main aspects:

(i) Single or a few default cases, as required, are not available.

(ii) The computing environment at the HLT farms are likewise the Grid, highly sophisticated.

The second, analyzing the impact of structural changes in the computing environment, could basically be addressed, but has currently no priority. The first, can partially be addressed by not only splitting use cases, but also the software into parts for further investigations. It also simplifies tracing performance issues back to source-code locations.

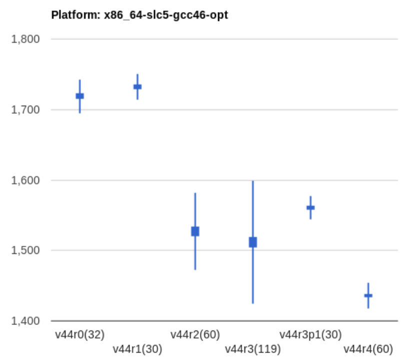

Figure 4. *nix*

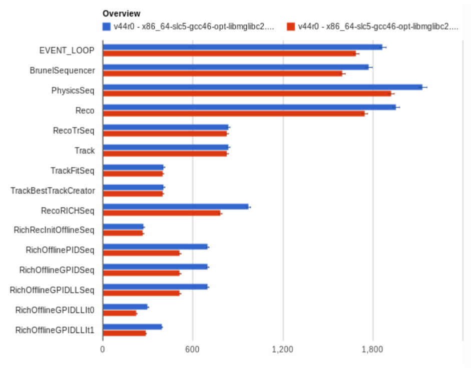

Figure 5. *nix*

3.8. Profiling accuracy

4. Performance analysis

The core of the LHCb performance and regression (PR) framework is its customized analysis platform based on Django [5]. The backbone of the framework is based on python which speeds up development while keeping a certain amount of flexibility. Additionally processing intensive tasks can be performed on the server-side, while keeping the interface quick and smart.

4.1. Web based analysis

4.2. Detailed analysis

5. Conclusions

Using a customizable platform to collect and summarize profiling results enables the LHCb collaboration to focus on important places in Gaudi algorithms during the refactoring time and beyond. This is organized in a way that takes additional tasks away from developers and simplifies the profiling to introduce a certain level of automation, e.g. for performance validation before a new release. It permits an arbitrary level of flexibility due to including new profilers and to monitor new software performance and quality values. The web front-end simplifies the task of monitoring the general performance of the Gaudi frameworks applications.

References

[1] G. Corti, M. Cattaneo, P. Charpentier, F. Markus, P. Koppenburg, P. Mato, F. Ranjard, S. Roiser, I. Belyaev and G. Barrand, *"Software for the LHCb experiment"*, IEEE Transactions on Nuclear Science, vol. 53, nb. 3, P.1323-1328, 2006

[2] M. Frank, C. Gaspar, E. v Herwijnen, B. Jost, N. Neufeld, and R. Schwemmer, *"Optimization of the HLT Resource Consumption in the LHCb Experiment"*, J. Phys.: Conf. Ser. 396 012021, 2012

[3] A. Mazurov and B. Couturier, *"Advanced Modular Software Performance Monitoring"*, J. Phys.: Conf. Ser. 396 052054, 2012

[4] D. F. Kruse and K. Kruzelecki, *"Modular Software Performance Monitoring"*, J. Phys.: Conf. Ser. 331 042014, 2011

[5] "Django is a high-level Python Web framework", url: https://www.djangoproject.com/

[6] "Jenkins, An extendable open source continuous integration server", url: https://www.jenkins-ci.org/