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 for the LHCb and other experiments based on Gaudi. In the upcoming years a big refactoring effort is planned to introduce features like vectorization, parallelization, 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

To assure that software performance of a large-scale framework keeps its preestimated performance boundaries, regular and detailed profiling is indispensable. Since software is not going to speed up anymore with each new generation of cpu's, profiling tools to trace back hotspots and to discover interesting regions to optimize the cpu's exploitation are becoming more and more important. But profiling was not an integral part of the software development procedure of the LHCb experiment for the first period of the LHC run till 2013. line-number-modeProfiling remained an unsystematic task which each developer or collaborating development group conducted on its own. Profiles were thus only of temporal value, incomparable and often unreproducible. This paper is not going to evaluate the benefits and drawbacks of different tools, but wants to enable the collaboration to systematize profiling and to make progress comparable and more visible for the upcoming refactoing face to make the source-code ready for the second big LHCb run. Important objectives to develop a framework for regular systematic profiling are, to be aware of comparability, reproducibility and automation of performance validation.
This paper is organized ...

2. LHCb computing

2.1. LHCb software

The software frameworks used in the LHCb experiment are based on Gaudi [1]. Gaudi is a framework using generic and object-oriented features of C++ for processing intensive tasks and python for configuring and structuring modules, which we call algorithms. 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 a large scale frameworks like the applications from the LHCb experiment. Multiple profiling measures have been implemented in the Gaudi framework using the Auditor-Service.
Timing information from the operating system's process information are collected using the TimingAuditor and 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 Amplifier which can be added using the IntelAuditor. Another strategy is to collect information from the PMU of a modern CPU architecture 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

Two important aspects must be considered as crucial for systematic profiling. It first must be frequently repeated and second it should be comparable to allow regression analysis. For this purpose profiling must be limited to small number of default cases using reference data to avoid the profile to vary because of a differing amount and size of physics events. This way differences between two software revisions can be examined on changes in their code. On the other side, changing events could be used to evaluate the performance of the software if other types of events for upcoming data-taking periods are expected.
Hence the LHCb PR project has the following objectives:

(i) Automated triggering and execution for frequent runs. This implies to avoid influences between jobs on the same machienes and makes job-configuration, job-submission and a job queue necessary. This is solved by using Jenkins [6] executing scripts as trigger to perform profiling and as wrapper to make the setup for several profilers transparent.

(ii) Profiles must be reproducable to be able to compare configurations. All activeties with changing configurations and all option files used are predefined and archived. Since Jenkins is missing the possabillity to be archived, the regular backup was put into a reposatory to be able to trace back changes in the jobs configuration.

(iii) Comparable, to trace back changes in the profile with changes in algorithms or in sorce-code locations. Hence it is also necessary to be conducted on library- and function level, what makes a detailed analysis necessary as soon as changes are observed. Profiles of a same kind become comparable by statistical analysis of multiple runs of a same kind of configuration.

(iv) Flexibal in including new profiling tools.

(v) Web supported brief analysis and detailed analysis of collected data. Means that many of the collected data of the several profilers will be stored for further analysis. Important information can be collected by parsing the profilers reports and pushing them onto a central database.

Systematic profiling should not only trace back hotspots and drawbacks of changes in code, but shell help to obtain an understanding which techniques, tool for optimization, or changing platforms are having a higher impact on saving computing resources.

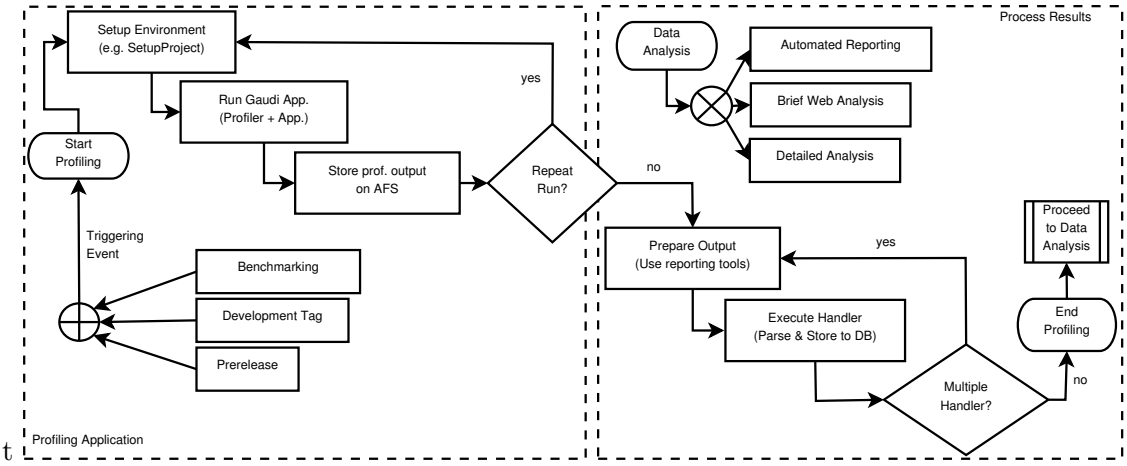

Figure 1. *nix*

3. The LHCb PR framework

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.

3.1. Structure

- Job distribution and triggering
- Job execution and profiling
- Data collection
- Web-based Analysis platform
- Test definition

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

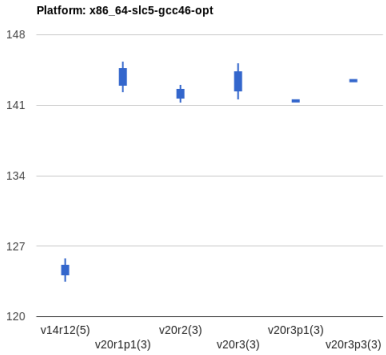

Figure 2. *nix*

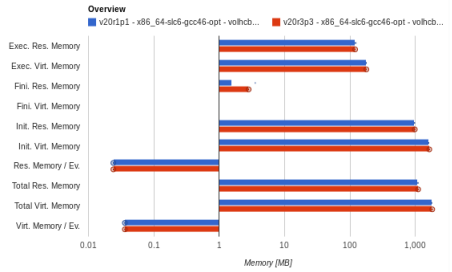

Figure 3. *nix*

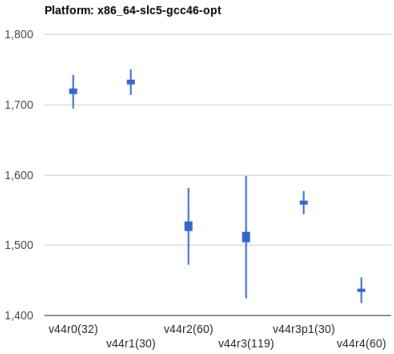

Figure 4. *nix*

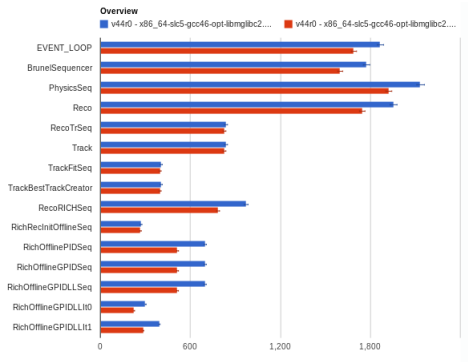

Figure 5. *nix*

execute the test cases and stay flexible for individual configuration. To improve the configuration 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. Unfortunately the HLT software Moore can not simply be reduced to a view most common default cases, what makes it more difficult to trace back performance issues that way as it would affect production. ...

3.5.1. Use case Reconstruction

3.5.2. Use case High-Level-Trigger

4. Performance analysis

The core of the LHCb performance and regression (PR) framework is its customized analysis platform based on Django [5].

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, Journal of Physics: Conference Series, vol. 396, no. 1, p.

[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/ 012021, Dec. 2012.