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 therefore 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 re-engineering and which kind of optimization could lead to final success.
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

In large scale software solutions of high energy physics, performance is critical for efficient result extraction. Plenty of profiling tools are available and must be integrated to trace performance already from an early state in development. But often software is more evolving because of considerations of maintainability, usability, flexibility, due to necessary features implementation or urgent bug fixes, than due to performance optimization. Often it appears, that such

information are not clearly and reliable available to determine the importance of performance optimizing measures. Hence, performance has less or no priority, also because there are no information about certain software behavior at all. The LHCb performance & regression (PR) project tries to address this issue for the LHCb experiment at CERN to allow more intervention which relies on performance information.
During its development phase, the LHCb software was constantly optimized; the profiling was however in 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 software performance, nor of the current baseline. With a refactoring of the code under way it is therefore now necessary for LHCb to put systematic profiling tests in place, in order to ensure that there is no performance degradation.
This paper describes the LHCb PR framework, to profile the experiments software and to display results in a user friendly manner. This paper describes an overview about LHCb software and the objectives of the LHCb PR project in chapter 2, the work-flow with some implementation aspects in chapter 3 and briefly goes into upcoming challenges in chapter 4.

2. LHCb computing

2.1. LHCb software

The LHCb experiment software is based on Gaudi[1], a C++ framework using generic and object-oriented features of C++ for computing intensive tasks, and python for configuring and structuring modules (algorithms, tools...). The application manager executes consecutive an abstract series of 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 Moore, Brunel, DaVinci, Gauss, Boole and others. Moore is the implementation of the High-Level Trigger (HLT) to decide weather event data will be stored or not, Brunel is responsible for the offline reconstruction of tracks, DaVinci is the physics analysis framework, 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 out of the computing resources accessed in the Worldwide LHC Computing Grid (WLCG), in Cloud Infrastructure and in the HLT farm located at the experiment. Additional, virtualization is used and just recently volunteer computing has been added. Some 100k CPU's are involved in the data processing.

2.3. Integrated Profiling

In HEP computing it is a common method to measure performance via throughput (events processed per time unit). Thus the performance analysis is focused on the time linear and not the time constant part of processing. 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 AuditorService [2] which provides an interface for executing code between events or algorithms processed.
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 Memory- or MemStatAuditor for changes in memory as soon as they appear. Recent work [4] conducted by Mazurov and Couturier shows how to improve precision in profiling the event-loop using instrumentation routines of Intel's VTune^{TM}Amplifier API, 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 usage and issues such as cache-misses, branch-misprediction and more as done by Kruse and Kruzelecki [5] 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 sparsely 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 and to trace back changes in performance to source-code locations. For a series of regular profiles for several platforms, profiling must be limited to a small number of default cases and a fixed set of reference data. Reference data are also important to avoid variance due to different types of physics. This way differences in execution behavior between two revisions can be examined and traced back on changes in related source-code. On the other side, changing the reference set of physics events could later on be used to evaluate the needs of computing resources for upcoming data-taking periods and changing recorded event information.
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 support brief 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 by trying to use supporting tools as much as available.

(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 project 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, Jenkins [7] a continuous integration system is used to configure, trigger and submit test jobs and finally Django [6] to support the web application to quickly visualize and propagate results.

3. Work-flow and Implementation

The LHCb PR project provides support to conduct systematic profiling. The work-flow of a test execution is summarized in figure 1, where Jenkins is used to configure, trigger and schedule test jobs before profiling. While execution, a profiler specific wrapper is downloaded and used to hide the profiler specific call for Jenkins and to setup the test environment for multiple executions. While execution, profiler information is gathered on a remote network file system for later detailed analysis. In the end, data handlers to parse reports and collect results are called, which are defined for distinct types of profiles and some additional job information like

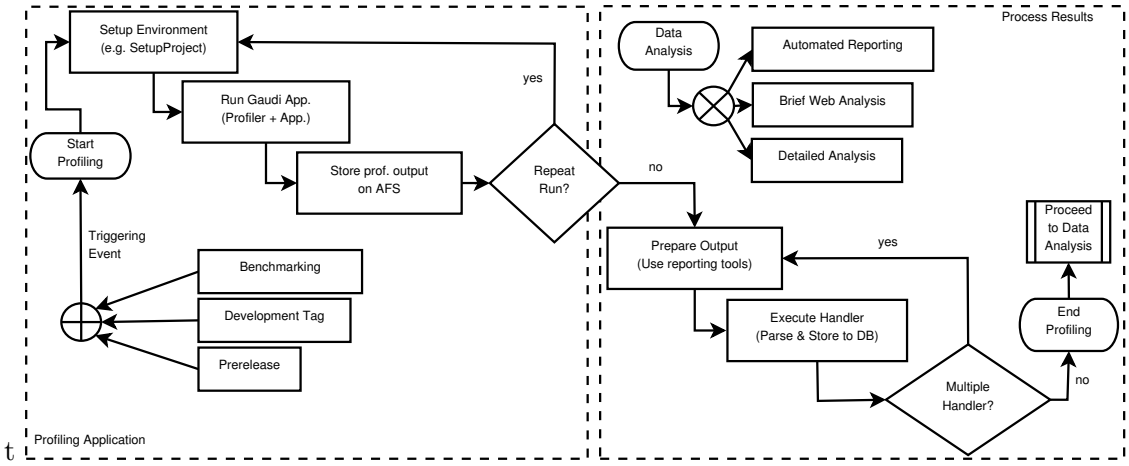

Figure 1. *The profiling procedure includes three stages. First, the test definition, triggering and submission. Second, the run execution, repetition and data collection by using data handlers to parse profiling reports and send core information to the database. And third, quick access to gathered profiling results by generating summary reports, using a web application for quick access to compare the general performance or to access detailed profile collected during the run.*

comments, files and classes of setups. Depending on the desired information, profiles can be filtered or combined with other available information. Then, results are hooked onto a SQL database. In the end the LHCb PR web analysis framework facilitates profile comparisons and helps detecting anomalies in performance.

3.1. LHCb PR framework

The core of the LHCb PR framework is a customized analysis interface based on Django. The backbone of Django is based on python to speed up development while keeping a certain amount of flexibility due to a variety of auxiliary modules available. Additionally, processing intensive tasks can be performed on the server-side while keeping the interface quick and smart.

LHCb PR can ruffly be divided into two main aspects, navigation and visualization. Plotting graphs is current done by using visualization capabilities of the ROOT framework and its python interface pyRoot or using Google Charts. Different kinds of analysis are referring to different ways of visualization in which data attributes, as generic items for performance information, are shown. For instance, the Basic-Analysis shows the distribution of attributes, the Trend shows the progress of attributes cross versions or revisions and the Overview-Analysis about several attributes between a few versions or platforms. Attributes can be runtime, resident memory, possibly lost memory or even more complex software performance or quality metrics.

For a top-down analysis, the Trend has been added to observe changes cross revisions and versions to figure out progress in performance. Significant changes become immediately conspicuous, but requires single attributes, e.g. one of many algorithms, to be tracked. A better observation of several attributed is allowed by the Overview. Both show entries with their statistical variance, but to get precise information about the distribution around their average, the Basic-analysis is indispensable using ROOT histograms.

To navigate easily through data, results can directly accessed by their category, a tuples of job description id, platform and host. To select categories, a generic selection menu allows to specify categories for comparison. The generic menu is amended by customized part to specify profiling groups (profiling information), attributes or filtering options important to the specific visualization. Data access is also available on job level via a job table accessible through a table

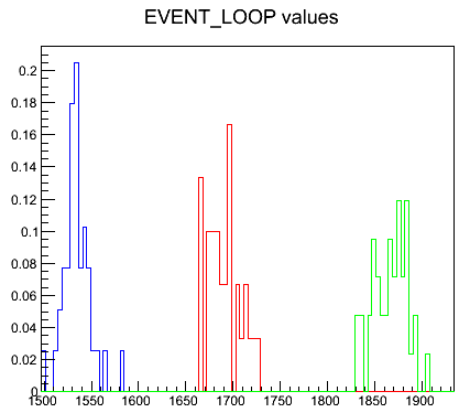

Figure 2. *Runtime distribution of a Brunel version using different glibc's ([ms] per event) of the event_loop attribute for different math libraries. (blue) Intel, (red) elder libm, (green) updated libm of a new glibc version.*

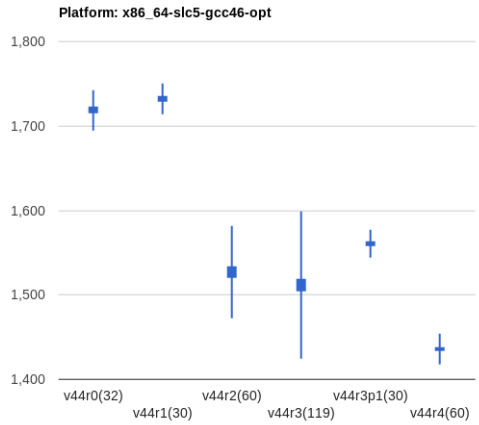

Figure 3. *Trend analysis of Brunel to monitor changes cross versions.*

of different test configurations or categories.

3.2. Job distribution and triggering

To facilitate regular profiling in a series of equal tests to permit statistic evaluations, Jenkins helps to manage the job distribution to test hosts. It makes the inclusion of further hosts simple and prevents interferences between multiple jobs running on the same machine by avoiding over-commitment. Test configurations can be prepared by creating parametrized jobs in the Jenkins interface. Test specific preparations, e.g. compiling packages (software modules) with specific compiler flags, can be added depending on the use case.

Despite configuring jobs, cron jobs in Jenkins can be used for triggering a profiling procedure within the release- or build cycle. This way, from triggering to execution and data collection, no human intervention is necessary, what simplifies data collection and opens the way to point out reliable and significant changes in the software profile.

3.3. Data collection

Data collection can be handled in three different ways, first one can segregate information from results to be stored in the database, second, files which contain performance information can be made available for downloading on the web-interface and third, bigger profiles can become available on remote storage. Segregating information from different profilers can be addressed by several parsers or data handler for each different profiling report. To maintain flexibility for many kinds of different profiling tasks, the post-processing of profiling must prepare parsable reports. The LHCbPR handler project attempts to collect these data. It further transfers information about the other, not parsed, profiling data for further examinations.

3.4. Test cases

Use cases are important to trace performance changes back to the evolving algorithms. Test cases shell base on default use cases for best approximations to the production usage. Still, the sophisticated computing environments can influence runs on a non-predictive non-deterministic way, which hampers conclusions from regression analysis. Currently, the highest priority is to find software related issues, that can be addressed or have to be taken into account for upcoming

decisions in resource allocation.
Default cases for Brunel have been quickly evaluated, defined and are running now on a regular basis after each successful build. LHCb PR has demonstrated its importance already by observing simple timing information obtained by the TimingAuditor. Significant changes become immediately clear by observing the trend information cross versions as shown exemplary in figure 3. After observing performance degradation further tests using the IntelAuditor have shown, that performance changes were once originated in external math libraries, as shown in Figure 2. VTune^{TM}can now be used for regular runs to visualize runtime spent in libraries and algorithms. 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 using a top-down analysis. The HLT brings further complications for the PR project in two main aspects:

(i) The computing environment at the HLT farms are highly sophisticated with processes cloned to run multiple processes on the same job, with events coming ...

(ii) Single or a few default cases, as required, are not available because of constantly changing trigger configuration keys (TCKs), which define the algorithms involved.

The first problem, as discussed, can not fully be addressed. First, more default cases can be used for a closer approximation and second, a standard configuration can be defined to make the analysis, as applied for Brunel, available. The second problem, can partially be addressed by not only splitting use cases, but also the software into parts for further investigations. This would also simplifies tracing back performance issues to source-code locations to help particular software development efforts.

4. LHCb PR and beyond

4.1. Profiling accuracy

In the current state, the web-interface does not provide a resolution beyond algorithm level. More detailed information are currently only accessible with the collected profile information profilers like Valgrind or VTune. The resolution of the interface is limited for two good reasons. First, developer shell be provided with profiles but competing with the variety of visualization tools of highly sophisticated profilers is unreasonable difficult and second, a tremendous amount of data that would have to be collected and stored. Still it could be reasonable to increase the resolution to function level. This becomes at latest with Moore evident, because algorithms are not executed in a strict order and for access to certain services lazy initialization is used. For instance tracking in the vertex detector is started by the first algorithm which requests these information and later on data can directly be accessed. This makes algorithm vary in their profile, what can currently not be reasonable traced back from the web analysis.

4.2. Complementary information

Due to the issue of the latter paragraph, the unordered execution of algorithms need to be addressed. This can be done by adding further complementary information, like a function call-stack for each algorithm to give a much higher resolution for finding anomalies in profiles than currently available, and it would make profiles of algorithms more comprehensive.
Another upcoming task is that if once a test case is fully implemented to be analyzed by several profilers in distinct runs, and if much more general information are available, these information could also be used to be correlated to each other. This way it would be possible to find the impact of design decisions and to validate existing methods or new concepts. Then open questions could be answered as knowledge base for further developments.
At the moment measurements like CPI, cache-misses, branch-misprediction and others are often mentioned, but can barely be correlated to real performance influences. Questions like how much exactly the call-stack depth influences runtime by occurring more cache-misses, and how

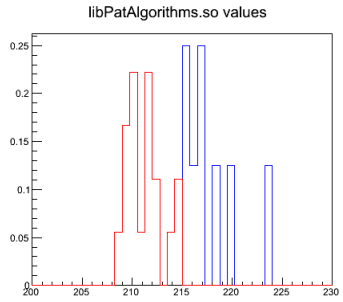

Figure 4. *Comparing auto-vectorized library in Moore again unvectorized version.*

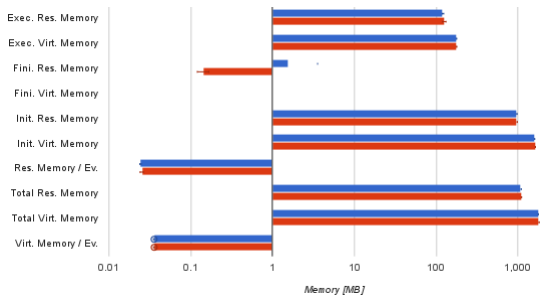

Figure 5. *For some application memory consumption is crucial. An overview of several memory information to observe differences between version.*

does this influences current R&D project like Gaudi-Hive by sharing cache among threads could then be emphasized by concrete numbers.

5. Conclusions

Using a flexible, extensible and customizable platform to collect and summarize profiling results enables the LHCb collaboration to systematically compare profiles. The LHCb PR project has already demonstrated to be highly valuable. Performance due to changing software and advancing technology could be observed, examined and appropriately be addressed. Since implementing instrumentation is in many aspects already performed, since many profilers can be applied for data collection and since Django and Jenkins is reducing the necessary work to set up a performance monitoring system, the effort is absolutely affordable for large scale projects as those from the LHCb collaboration. Still, improvements have to be discussed and put into place, but due to the easy adaptivity to the web interface and testing framework, more and more fields for application, like including software metrics into testing and analysis become tangible.

References

[1] G. Corti, M. Cattaneo, P. Charpentier, M. Frank, 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] P. Mato and others, *"Status of the GAUDI event-processing framework"*, Intl. conference on computing in high energy and nuclear physics, Beijing, China, 2001

[3] 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

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

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

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

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