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

Ben Couturier [1] Stefan Lohn [1] Emmanouil Kiagias [2]

[1] LHCb, Department of Physics (PH)
CERN, European Organization for Nuclear Research
CH-1211 Genéve, Switzerland

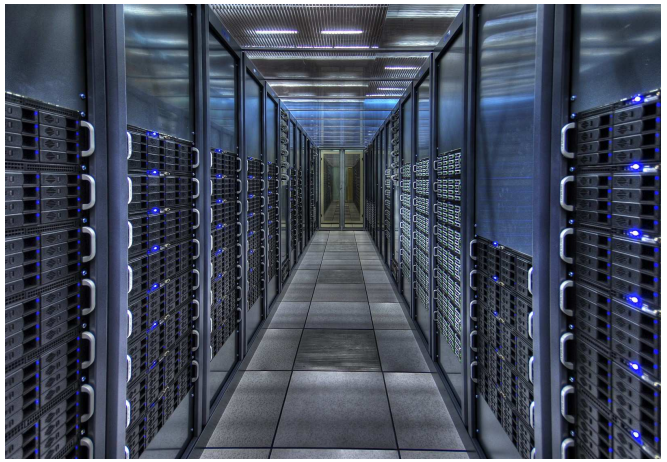
[2] University of Athens
GR-1211 Athens, Greece

October 14th, 2013

Introduction - 1

Computing environment

- **Distributed computing:** Worldwide LHC Computing Grid, Cloud technology, HLT computing farm.
- **Parallel processing:** Multi-Threading, Multi-Processing and GPU processing is advancing.
- **Virtualization:** Used in Grid and Cloud.
- Recently **volunteer computing** using BOINC.



Photograph: Roger Claus CERN, Date: 07 Aug 2010

LHCb software frameworks and applications

- GAUDI, the *general data processing framework* of LHCb, is a **template C++ framework** to provide basic services and tools. Gaudi *applications are using Gaudi to implement algorithms* (software modules) to perform the event processing.
- Applications which are using Gaudi are:
  - Moore, High-Level-Trigger (HLT) framework.
  - Brunel, offline-reconstruction.
  - Gauss, monte-carlo simulation framework.
  - and others.
- Gaudi is **used by LHCb, ATLAS and other non-LHC experiments**.

Introduction - 2

- Software is **continuously evolving**
  - Continuous source-code development.
    New or improved algorithms, services or tools.
  - Integration of external software.
    E.g. transition from Root 5 to Root 6 will integrate Cling.
- **Software refactoring** during Long Shutdown (LS1) and beyond:
  - HLT splitting in HLT1 and HLT2.
    More filtering by increasing quality and precision.
  - Introducing GaudiMP for moldable job submission in the Grid. [1]
    Reduce memory consumption by exploiting parallelism.
  - Redesigning Gaudi for Gaudi-Hive. [2]
    Introducing multi-threading to increase CPU exploitation.
- **Advancing technology**
  - Performance impact of new Platforms.
    Changing OSes, New hardware architectures.
  - New compiler optimizations.
    Auto-vectorization, auto-parallelization, profile guided optimization.

Always the same questions

- **Is it worth it?** *How is the expected and final performance impact?*

Integrated Profiling

Profiling Information

- Integrate profilers into your **environment**

  e.g Valgrind accessible from AFS, TCMallog usable by flag, IgProf usable by flag, ect.

- Insert routines for **instrumentation**

  GaudiAuditor can be used to insert routines within event-loop.

- Profile your **development progress**

  Everyone is required to profile his work.

Many implementations have been conducted successful:

- Timing- and MemoryAuditor (using process info.)
- Precise timing per function, algorithm, library and more using IntelAuditor [4].
- HW-Event based profiling (cache-misses, branch-prediction, ...)
- Sophisticated memory analysis (shadow memory, heap analysis)
- Creation of call-graphs

Important

Integration of profiling tools to be accessible throughout nodes **and instrumentation is an important step** for high-performance software, and already performed for the Gaudi-Framework.

Systematic Profiling - 1

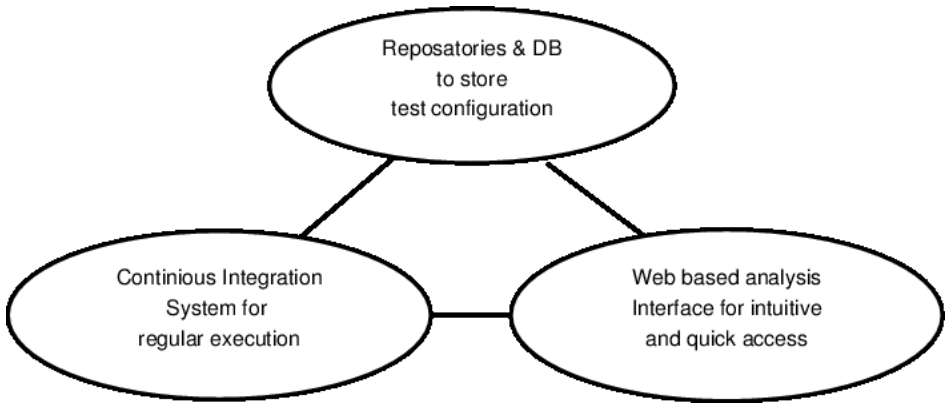
Problems

Usage of profiling must go above the temporal use! But:

- Reduced use cases and limitations of tested code regions, makes them **incomparable**.
- Many are **unreproducible** due to unknown test configuration.
- Apply only for uncommon or **unrepresentative** use cases.
- Are **labor intensive** if done regularly and precise.

Idea

- **General use cases** give approximated profiles to become comparable.
- Processing on a limited set of **representative reference data**.
- Test configuration must be stored to become **reproducible**.
- **Regular execution** in a serious of same test runs to increase precision and keep track on changes.
- **Facilitate analysis** by unifying profile results into a single web application.



Systematic Profiling - 2

Solution

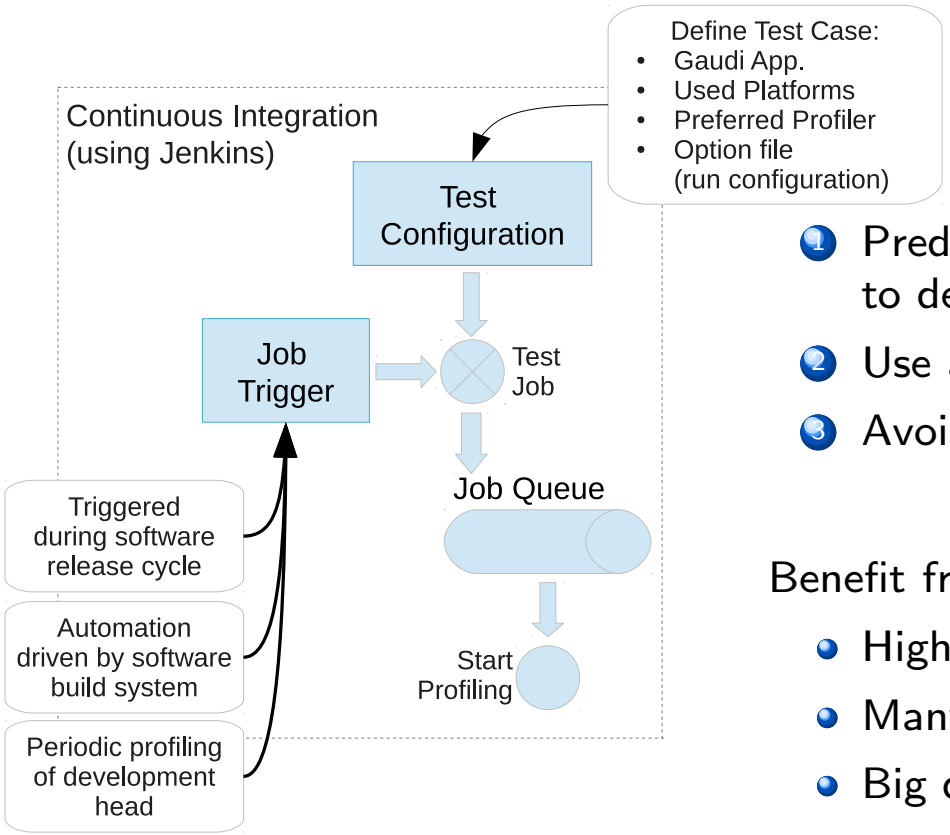
The **LHCb performance & regression (PR) Project**
A system to provide tools for test execution, collecting core information and support for analyzing profiles.

Major goal: *Observe general performance to estimate resource consumption and to find anomalies in these. Hence we perform a kind of top-down analysis.*

Technical requirements

- Flexible and extensible integration into the LHCb PR project of any profiling tool.
  Create and parse reports to store profiling results using data handlers.
- Quick access to general and detailed performance information of executed profiles.
  Customized visualization to browse results intuitive and comprehensive using a web interface.
- Reduce work by investigating generic ways of navigation and visualization.
  Use a productive web application framework, reuse visualization tools.
- Automation in case of regular profiling and warning generation.

Workflow of profiling - 1

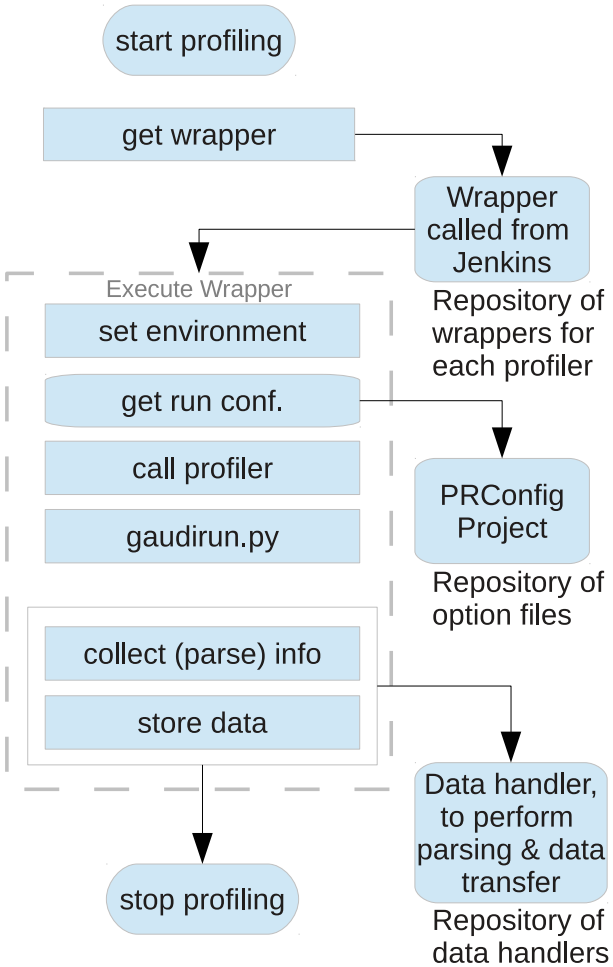


1. Predefine parametrized job configuration to define your test cases.
2. Use a job trigger for automated profile initiation.
3. Avoid overcommitting nodes (job queue).

Benefit from using Jenkins:

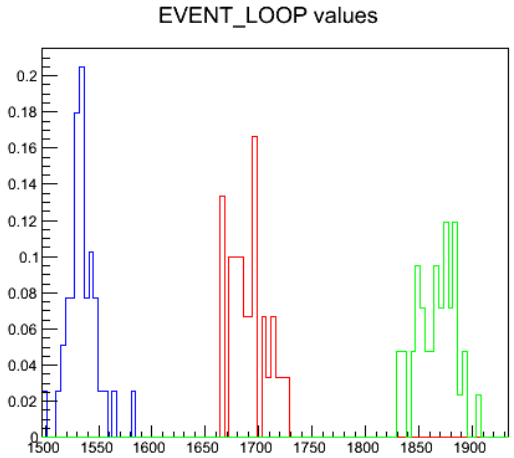
- Highly configurable.
- Many additions due to plug-in support.
- Big development and user community.

Workflow of profiling - 2

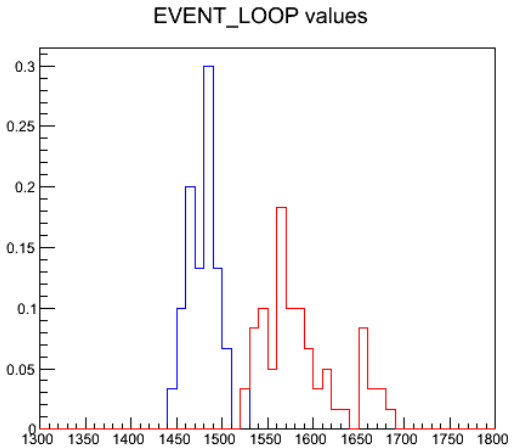


- Job configuration persists out of:
  - Parameter from test configuration.
    Environmental setup (like CMTCONFIG, MYSITEROOT, ext.) relies on test configuration.
  - Run configuration.
    Option files from PRConfig.
  - Profiler dependent setup.
    Choice of used profiler (wrapper) & handlers.
- Wrapper perform:
  - Setup of environment
  - Call of profiler and execution of test job.
  - Choice of data handlers for collecting core information.
- Handler perform:
  - Parsing information from profiles.
  - Hook data to storage element for later import to DB.

Tracking General Performance - 1



(Blue) peak shows runtime per event in millisecond in Brunel v44r0 with intel's math library, (red) libm of glibc-2.5.65 and (green) libm glibc-2.5.107.



(Red) shows runtime of Brunel on SLC5 vs. (blue) SLC6.

Know your position

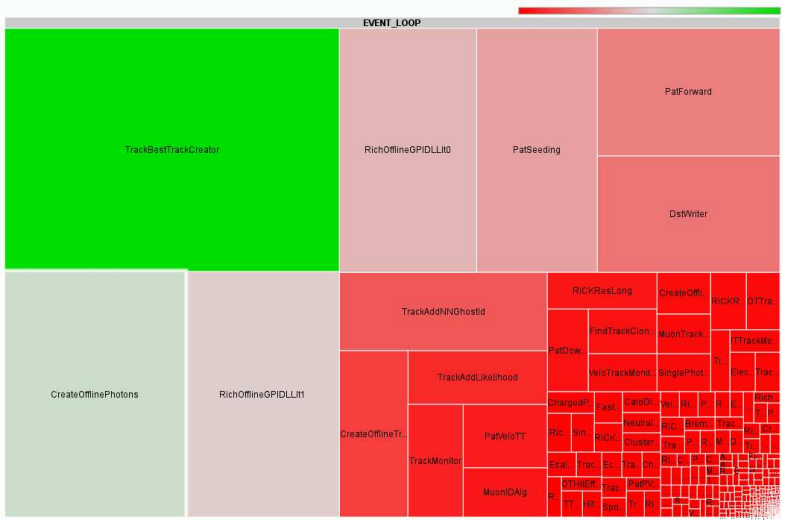
For tracking general performance it is important to know about the **overall performance of a specific test case** and to compare it cross single differing parameters like version or platform.

BASIC-Analysis: Shows attribute distribution

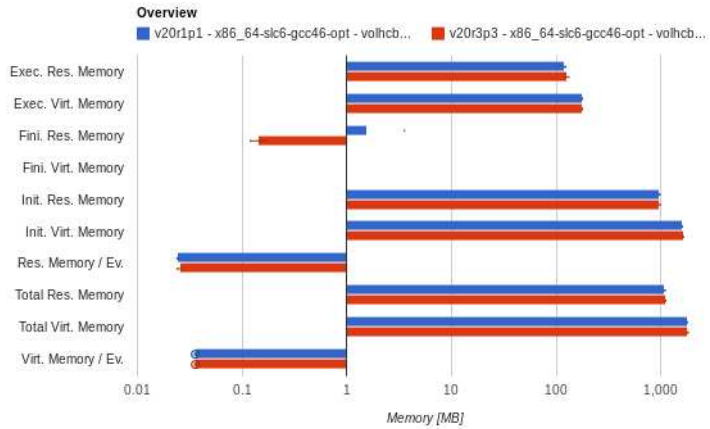
To visualize values of an attribute like event-loop (runtime per event) for multiple runs, the BASIC analysis shows the distribution. This can be compared with other run-configurations.

Visualization using ROOT diagrams.

Tracking General Performance - 2



Overview of runtime proportional to others.



Comparing memory for Moore between versions.

Keep an eye on Modules

Plenty of modules (algorithms/libraries) are profiled. **Single changes can affect all Modules** in runtime. Profile *regular also with minor software changes* to find the origin.

OVERVIEW-Analysis: Overview about attributes

Attributes can be specific modules or categories of performance, like resident memory. Plenty of attributes are available. The **overview analysis gives a quick impression** and makes comparisons between few versions and platforms possible.

Visualization using Google Charts.

Use case High-Level-Trigger

Structural Problem

Some further structural problems have been found by profiling Moore.

1. TCK's (Trigger Configuration Keys) configure different set of algorithms, what makes the **definition of a default use case difficult**.
2. **Differences in same Algorithms** are expected. The first calling algorithm performs creation of certain objects. Algorithms do not have a fixed execution order.
3. **Runtime environment** in HLT computer farm **differs** from execution with test jobs on a local machine. Indeterminism in complex environments makes profiles varying and be less expressive for source-code performance.

- 1) & 3) increases the **differences between test-cases and production**. Differences rely partially on **indeterministic influences** which make performance comparison less accurate.
- 2) can be addressed by segregating **call-stacks for each algorithm**, but uses a lot of space. E.g. VTune is not delivering such information in great details by command-line.

Conclusions & Perspective

Conclusions:

- LHCb PR has shown to be **easily usable** to point out **core issues** of performance.
- Benefit from **automated profiling** for a series of regular profiles allowing statistical analysis.
- **Flexible extensible framework** to integrate further tools for profiling.
- Using **clear visualization** with support to propagate results **assures improvements** in performance.
- **Don't reinvent the wheel:**
  - Jenkins has shown to be highly valuable.
  - Django is speeding up adaption in the web-based analysis interface.
  - Use available profilers, that has demonstrate their usability.

Future perspective:

- **Function-call stack as complementary** information should be accessed and integrated into the web interface.
- **EBS** is currently not integrated but **has shown more and more to be effective** to find software performance issues.

References


Nathalie RAUSCHMAYR, “Preparing the Gaudi-Framework and the DIRAC-WMS for Multicore Job Submission”, Amsterdam, 2013


Danilo PIPARO, Dr. Pere MATO VILA, Benedikt HEGNER, ”Introducing Concurrency in the Gaudi Data Processing Framework“, Amsterdam, 2013


Vijay Kartik SUBBIAH, Niko NEUFELD, “Measurements of the LHCb software stack on the ARM architecture”, Amsterdam, 2013


Alexander MAZUROV, “Advanced Modular Software Performance Monitoring“, New York, 2012

Questions?