The investigation of CMS software through an automated performance measurement and analysis system.

V. Innocente

CERN, Geneva
**E-mail: vincenzo.innocente@cern.ch*

R. Moser

Department of Computer Science, ETH Zürich
**E-mail: mail@robinmoser.ch*

D. Piparo

CERN, Geneva and Dipartimento di Fisica Università degli Studi di Milano Bicocca
**E-mail: danilo.piparo@cern.ch*

L. A. Tuura

Northeastern University, Boston, USA
**E-mail: lassi.tuura@cern.ch*

A set of new tools was developed to measure the performance of the CMS offline analysis software. We describe the procedure adopted to benchmark the CMS software together with the tools deployed to configure the CMSSW application, produce the performance profiles, create the performance reports and publish them.

Keywords: CMS; Software Performance; PerfReport; Python for Configuration

In this paper we describe the procedures and the tools developed and deployed to measure the CPU, memory and I/O performance of typical CMS offline software (CMSSW) applications.[1] In section 1 we show how a CMSSW application can be configured using Python to suit the requirements of a performance measurement. Section 2 is devoted to the description of the profilers exploited. We depict the processing procedure of the reports through the PerfReport[2] tool in section 3. In section 4 we characterise the PerfReport3 web server, the product used to publish the reports. The conclusions are exposed in section 5.

1. The profiles production: PyReleaseValidation

The CMS software release validation (RelVal) system provides a set of standard event samples to test basic features for each new release. Performance measurements share with release validation several aspects such as code coverage, specific configurability, the ability to run either for few events or as a large production jobs. The set of static configuration files, such those used by the standard RelVal team, is unable to provide the flexibility needed for the performance measurements. An alternative configuration system, based on python scripts, was therefore developed.

The applications used to benchmark CMSSW are today configured by means of PyReleaseValidation, a tool written in Python [a] which, not only can reproduce exactly all the functionalities of the traditional RelVal, but brings in new features desirable for the benchmarking goals. Among all the options available,[3] the user can vary within the commandline number of events, select the input and output files, perform a full chain from Montecarlo generation to reconstruction or execute only one of the steps in the middle or select the algorithms of a sequence.

2. The profilers

CMS has decided to mainly rely on the two profilers, IgProf and Callgrind, for performance measurements. IgProf is a statistical performance and accurate memory profiler.[1] Its main strength is the small overhead it adds during profiling. In performance mode it works by sampling callstacks at regular time intervals. Complementary to IgProf, Callgrind, from the Valgrind family of tools [4,5], can be used to analyse short runs of selected algorithms with maximum scrutiny.

Callgrind usually records a summarised call graph in which each symbol appears as one vertex and the inclusive costs of calls are recorded along the graph's edges. As it can no longer be deduced how the costs distribute to single callstacks, this data is insufficient for a complex setting like CMSSW. Therefore, a patch called FCE (Full Callstacks Extension) has been issued for Callgrind, which embeds an IgProf-like full call tree data structure into the data collection framework of Callgrind.

[a] part of the CMSSW distribution under *Configuration/PyReleaseValidation*

3. Processing the reports

The large amount of "raw" information which profilers produce needs to be analyzed and presented to allow for quick identification of particular issues. PerfReport analyses raw profiling data stemming from either Callgrind FCE or IgProf and reports the results in the form of either a static HTML tree or of data written into a database for later representation in an interactive web interface (Fig. 2 and 4)

The profilers produce weighted call trees. Each node of such a tree corresponds to a callstack obtained by following the path to the tree's root. The amount of time or memory spent, used or leaked on that callstack is then recorded in the corresponding node. On such detailed input, PerfReport can perform sophisticated types of analyses. Built-in filtering capabilities include most operations naturally induced by such a tree structure. To remove unwanted costs, subtrees can be pruned (Fig. 1, top left). To distribute costs to callers, a subtree can be collapsed into its parent node (Fig. 1, top centre). Collapsing only one node into its parent will do the same while preserving calls made by the collapsed function (Fig. 1, top right). It's also possible to collapse sets of nodes both within and across levels. A description of any chain of these built-in filters is provided using a simple XML grammar, using among other things regular expressions on symbol names and context constraints to restrict the operation to certain branches.

PerfReport is designed so as to allow arbitrarily big profiles. It maintains the profile tree on disk and brings the parts needed for analysis to memory on demand (Fig. 1, bottom left). Some large analysis data structures are kept on disk as well.

When summarising, symbols can be partitioned into either predefined or user-defined categories and then aggregated. Aggregating into natural buckets like classes or namespaces is supported built-in. For this purpose, PerfReport needs to parse and dissect symbol names. It implements a parser component which correctly recognises the sometimes complex symbol names occurring in CMSSW. This component has also been contributed for use in future versions of KCachegrind[4] to allow for proper aggregation in case of C++ profiling targets.

As performance optimisation is a continuous rather than a one-time task, the developers have to keep track of the measurement data as the development evolves. PerfReport helps with this process by offering a diff-like feature to compare profiles stemming from different releases of the same target application (Fig. 1, bottom right). It highlights important changes in the performance numbers, endeavouring to automatically select what is signif-

icant and informative. Technical challenges in this field include accounting for nomenclature changes and presenting minimal sets of differences.

A threshold value can be used to limit how much data PerfReport writes into a database. The callstacks of which the profile is made up are then ordered by importance, according to how much resources are spent on each stack or its symbols. In this order, they are written into the database up to exhaustion of the allowance.

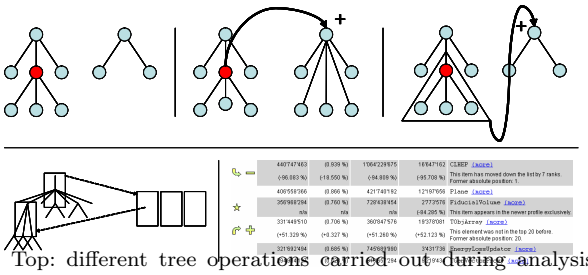

Fig. 1. Top: different tree operations carried out during analysis. Bottom left: disk buffering. Bottom right: regression view

4. Publishing the reports

Making the performance reports available to people is crucial. For this reason, Perfreport3 includes a webserver application based on Webtools technology.[6] This product is developed in Python and acts as an interface to the performance db. Once connected to the server with a web browser, the users can select the desired CMSSW version, describe their own filters and display options (Figure 2). A browsable html report is generated on the fly according to the preferences and every page is bookmarkable. The implementation is tailored to ease the introduction of new filtering criteria. The application is fast enough for the web product standards. A C++ Python extension which offers access to the PerfReport API is used to overcome the computationally heaviest aspects. The user can read a quick documentation in a sliding panel which appears clicking on the help button and quick information is provided by pop-up tooltips.

5. Conclusions

The PyReleaseValidation proved to be an essential component of the procedure to routinely produce performance report for each release. IgProf

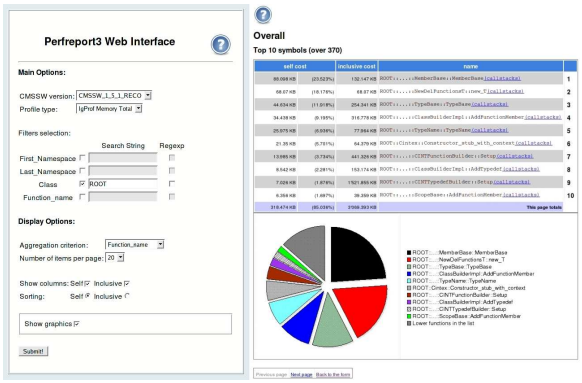

Fig. 2. Left: The web form to allow the user to make the selections. The first version of the interface already supports regular expressions and aggregation of symbols. Right: a report which is generated dynamically, fully browsable.

and Callgrind (FCE) have shown to be well-suited to collect performance data in a non-trivial environment as the CMS offline software. Turning the "raw" performance data into intellegible reports requires a sophisticated post-processor. PerfReport provides an easy configuration mechanism that allows to produce performance reports that are easy to read and tailored to specific use such as search for "hot spots" or the publication of condensed summary. The webinterface of Perfreport is a powerful approach to the issue of profile publication that exhibits high performance in terms of both speed and interactivity.

References

1. G. E. Lassi Tuura, Vincenzo Innocente, Analysing cms software performance using igprof, oprofile and callgrind, in *Proc. CHEP 2007*, (Victoria, Canada, 2007).
2. *Perfreport.* https://twiki.cern.ch/twiki/bin/view/CMS/SWGuidePerfReport.
3. D. Piparo, The cms ecal calibration at testbeam 2006 and search of the $\tau \to \mu\gamma$ rare decay channel, Master's thesis, Università di Milano Bicocca (2007).
4. J. Weidendorfer, *KCachegrind and Callgrind.* http://kcachegrind.sourceforge.net.
5. N. Nethercote and J. Seward, *Electronic Notes in Theoretical Computer Science* **89** (2003).
6. *Web Tools.* https://twiki.cern.ch/twiki/bin/view/CMS/DMWTDeployment.