Developing a tool for analyzing the jobs' performances on CRAB3

Luca Ambroz
-
Supervisors:
Dr. Andreas Pfeiffer
Prof. Daniele Bonacorsi
Dr. Giuseppe Codispoti

***Abstract*—In this report, the design and development of a tool capable of providing a quick and intuitive analysis of the measured performances in CRAB3 job submissions, on any Grid or Cloud resources, is provided. The tool has been tested with job submissions on Grid and Cloud infrastructures specifically designed for benchmarking.**

I. Introduction

There are two main reasons for developing a tool which can give feedback about the performances of the execution of tasks and jobs in CRAB3 [1] [2]. To describe them two scenarios will be presented. A computer scientist interested in the analysis of the efficiency of two different infrastructures - for example the Worldwide LHC Computing Grid [3] and any Cloud infrastructure - could easily execute various tasks on both of them, and then with two commands, obtain information on how both resources were exploited (informations that are complementary to those given by the Dashboard [4]). Whereas an analyst could use the same informations for different purposes. For example, he might desire a fast but comprehensive glimpse on how his jobs are performing. This procedure can be quite cumbersome and time consuming, thus there was the need of tool capable of extrapolating these informations from the log files of the tasks.

II. Design

The tool has been written using Python. In order to operate it, a user that has carried his analysis in a specific working directory (specified in the `crabcfg.py` file) for several tasks should retrieved the logs through the CRAB command [5] [6]:

```
crab getlog <working-directory>/<task-directory>
```

Then he simply needs to run the tool giving as argument the working directory:

```
python tasks_analyzer.py <working-directory>
```

and, at the end, the program creates a directory called `Tasks_Analysis_<working-directory>` containing both the informations about the single tasks and on overall analysis of the tasks.

III. Single task analysis

For each task contained in the working directory the tool performs several actions. Firstly, it unpacks the log files and stores them as .xml and .log files. Secondly, it retrieves informations from these files and produces a dump of them in a .csv file which can be then use for a more sophisticated analysis using packages such as R. The metrics which have been studied are:

- **CPU Time**: time the application spent on WN CPU.
- **Execution Time**: overall job execution time.
- **CPU Efficiency**: parameter which evaluates the CPU's efficiency as follows:

$$\text{CPU Efficiency} = \frac{\text{CPU Time}}{\text{Execution Time}}$$

- **Number of events**: total number of events per each task.

Thirdly, with the above metrics, it produces plots in .png format. Moreover, the graphs are stored in root files named `<task-name>.root` for further analysis exploration with ROOT.
As a demonstration of the features of this tool, in Fig. 1-10, the plots produced by analyzing four tasks submitted to a Cloud infrastructure in Bologna are shown. These tasks were used to compare the performance of a Cloud infrastructure to the Worldwide LHC Computing Grid and provide a proof of concept of the elastic extension of the CMS-Bologna Tier-3 Site on an external Cloud infrastructure, implemented on OpenStack [8]. A newly designed LSF [7] conguration was used in what it can be considered a "Cloud Bursting" of a traditional CMS Grid Site [9]. In principles, the colleagues in Bologna aimed to extend the Grid to a Cloud infrastructure whenever it was need, without a user could even realize that this procedure was happening. However, there are small differences between the two architectures and these can be detected thanks to this type of analysis tool.

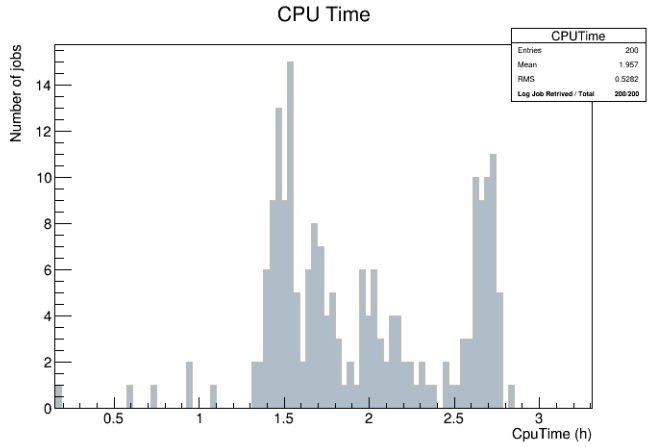

Fig. 1. Number of jobs as a function of the CPU Time for a workflow that is described in detail in the text.

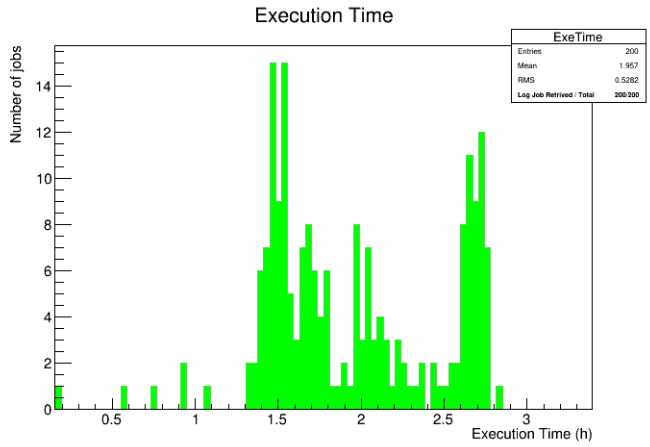

Fig. 2. Number of jobs as a function of the Execution Time for a workflow that is described in detail in the text.

The plots in Fig. 1 and 2 provide a general overview of the time of execution of the single jobs. With this type of information, an analyst can optimize his job submission exploiting his computing resources without having moments in which there are no jobs running. For example, take Fig.1 and consider this case. An analyst typically will have to submit many tasks consisting of many jobs each, and would aim to see them finished in the minimum amount of time. It is realistic that (s)he submits some jobs only, or a small task, first to figure out how much time it may take to run all tasks in the work queue. Running a small task and/or only few jobs may not be statistically significant, i.e. they may end up in underperforming Grid sites (thus overestimating the overall work completion time), or only few of them may succeed (thus yielding a not significant statistics of successful jobs to draw any conclusion). It is probably wiser to submit a realistic, medium-size task and use the tool documented in this project to analyze the actual CPU time and CPU efficiency patterns of all the jobs in such task. Consider for example that this is done and the streamlined analysis of its results allows to conclude that the vast majority of jobs in the test task complete in less than 8 hours running on 5 different (in performance, location, etc) computing centres. It will hence be straightforward to presume that also all the other tasks may do the same: as a consequence, having this insight at hand, a proper scheduling of the user's submissions in the next few days is possible, and ultimately the whole work completion time will be largely reduced with respect to a "let's submit radomly and see what happens" approach.

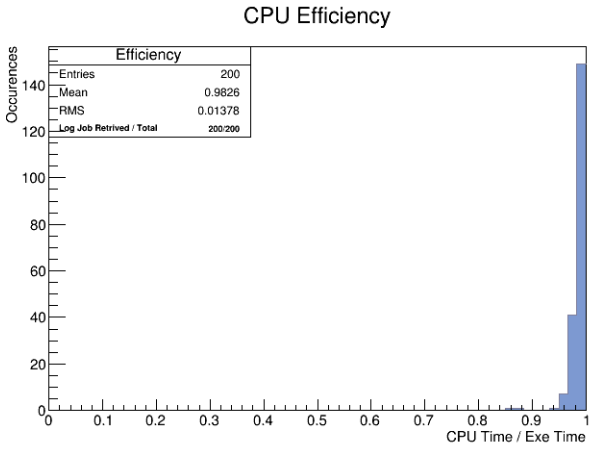

Fig. 3. Number of jobs as a function of the CPU Efficiency for a workflow that is described in detail in the text.

The plot in Fig. 3 shows if the computational resources are used efficiently or whether there are problems for certain tasks. Furthermore, this metric is fundamental to confront the performances of the Grid to any Cloud infrastructure. Indeed, in the context of the comparison previously described, one expects to obtain different peaks of CPU efficiency for the Tier-3 Cloud infrastructure in Bologna and the Grid. In the end, it is a fast way to evaluate two completely different and unrelated architectures.

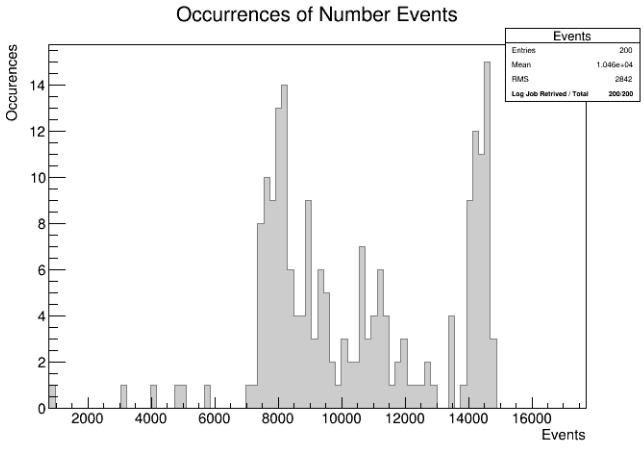

Fig. 4. Occurrences of the number of events for a workflow that is described in detail in the text.

It is also interesting to have a quick graphical way to monitor how many events have been processed - as from the info in the logs - without actually looking at the logs themselves but

by just parsing them and filling an histograms, as shown in Fig. 4.

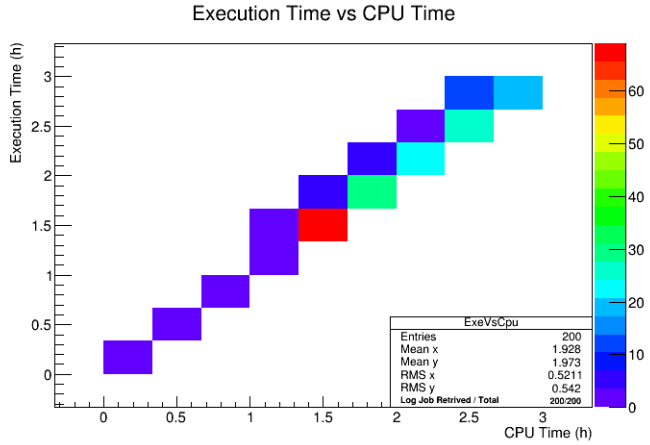

Fig. 5. Execution Time as a function of the CPU Time for a workflow that is described in detail in the text.

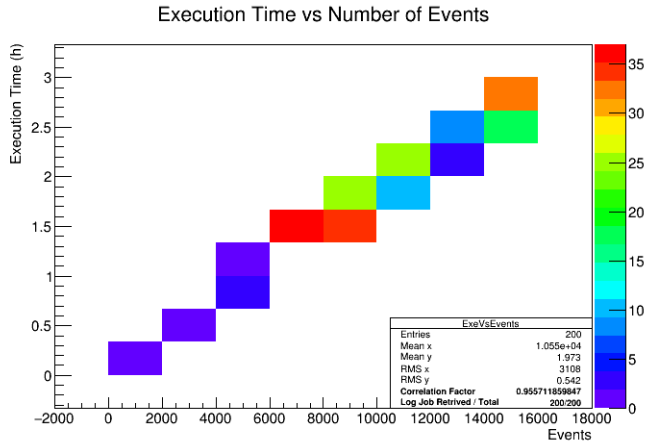

Fig. 6. Execution Time as a function of the number of events for a workflow that is described in detail in the text.

A feature which is highly desirable in this kind of tools is to been able to plot correlations between different observables. Thus, this tool can produce bi-dimensional plots of all the metrics discussed above according to the user's needs. The plot in Fig. 5 shows how the execution time is correlated to the CPU time. The graph in Fig. 6 looks for the degree of correlation between the Execution Time of each task and the Number of Events contained in each task.

IV. Multiple Task Analysis

So far, we discussed a per-task monitoring, in which all entries in a plot refer and belong to just one task. Another useful aggregation in CRAB3 is per Working Directory. In fact, typically, users need to submit plenty of tasks, some of which may become useless as the analysis proceeds, but some other may be a reference for all the subsequent tasks creation and submission. It is hence interesting, inside a unique working directory, to graphically compare - for example - the average CPU time (or other observables) *per task*, in a single *per working directory* plot.

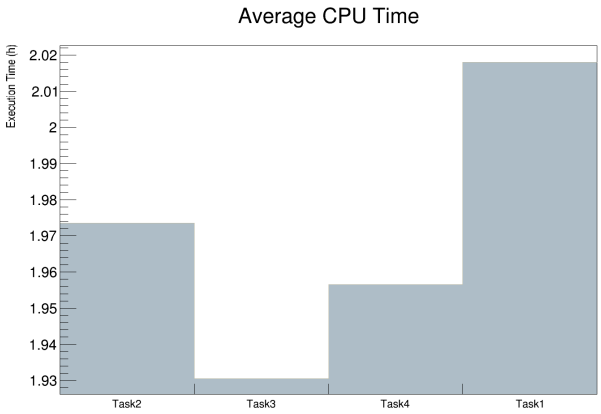

Fig. 7. Averages of CPU Time for four related workflows.

In Fig. 7, the averages of CPU time of different tasks are plotted. This may help to spot the tasks that use more CPU time in their jobs' execution.

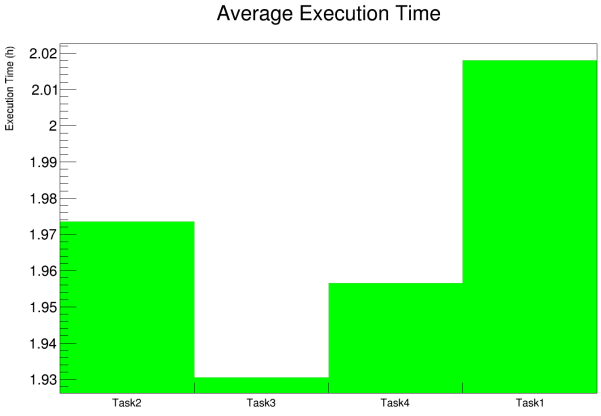

Fig. 8. Averages of Execution Time for four related workflows.

Analogously, in Fig. 8, the averages of Execution Time for different tasks are plotted. This could help identify which tasks take overall more time to finish.

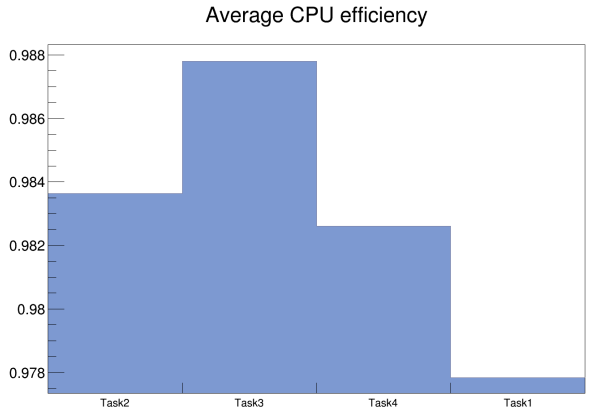

Fig. 9. Averages of CPU Efficiency for four related workflows.

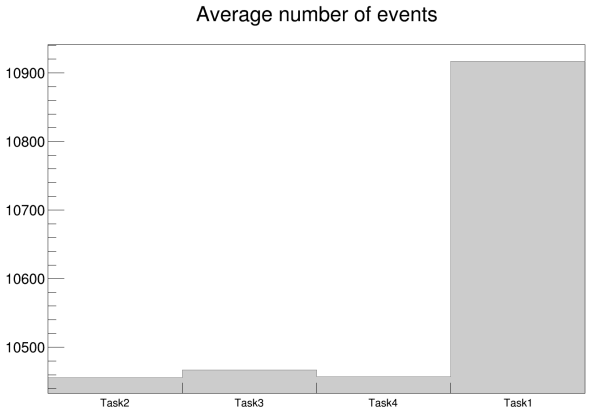

Fig. 10. Averages of number of events for four related workflows.

From the combined analysis of these plots it is possible to extract even more valuable informations. For example, let's consider the plots in Fig. 9 and 10. In Fig 9, Task1 has a lower CPU efficiency compared to other similar tasks. Now observing the average number of events of the tasks in Fig. 10, one can conclude that having to access a larger number of events might be the cause of the lower CPU efficiency. This has only been possible by observing the four tasks in parallel.

V. Job submissions for benchmarking

The tool has been tested with job submissions on Grid and Cloud specifically designed for benchmarking. The first tests of the tool were performed on six small tasks in which it was fixed the `NJOBS` to 10 whereas the `unitsPerJob` were progressively increased (1, 2, 5, 10, 20, 50). These tests were used to initially developed the tool. Once the reliability of the tool was reassured, the tool was tested with log files coming from the analysis of the Tier-3 Cloud infrastructure in Bologna containing up to 200 jobs. All the tests were based on the same workflow: an analysis task in the context of the fully hadronic top physics [9].

VI. Conclusion

The tool is suitable for analyzing various tasks with several hundreds of jobs without too much effort. In the future the tool could be extended for investigating the causes which prevents high values of CPU Efficiency (e.g. certain Grid sites with inefficient machines or see whether the data were retrieved through remote access). Furthermore, this tool has the potential to become a service for the users in CMS either as a tool-kit downloadable from GitHub or as a web application in which the user has only to upload his log files.

References

[1] D. Spiga et al., "*The CMS Remote Analysis Builder (CRAB)*", Lect. Notes Comput. Sci. 4873 580-586 (2007)
[2] Giuseppe Codispoti et al., "*CRAB: A CMS Application for Distributed Analysis*", Nuclear Science Symposium Conference Record N02.79 (2008)
[3] J. D. Shiers, "*The Worldwide LHC Computing Grid (worldwide LCG)*", Computer Physics Communications 177 (2007) 219–223
[4] The Dashboard project, http://dashboard.cern.ch
[5] Software Guide on CRAB, https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab
[6] CRAB3 online tutorial, https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCRAB3Tutorial
[7] S. Zhou, J. Wang, X. Zheng, P. Delisle, "*Utopia: A load sharing facility for large, heterogeneous distributed computing systems*", TechnicalReportCSRI-257, Computer Systems Research Institute, University of Toronto (1992)
[8] OpenStack, http://www.openstack.org/
[9] R. Di Maria, "*Elastic Computing on Cloud Resources for the CMS Experiment*", Master thesis (2015)