Use of DAGMan in CRAB3 to improve the splitting of CMS user jobs

To cite this article: M Wolf et al 2017 J. Phys.: Conf. Ser. 898 052035

View the article online for updates and enhancements.

Related content
- Key developments of the Ganga task-management framework
  I A Dzhunov, M J Kenyon, J Moscicki et al.
- Use of glide-ins in CMS for production and analysis
  D Bradley, O Gutsche, K Hahn et al.
- Use of the gLite-WMS in CMS for production and analysis
  G Codispoti, C Grandi, A Fanfani et al.
Use of DAGMan in CRAB3 to improve the splitting of CMS user jobs

M Wolf, M Mascheroni, A Woodard, S Belforte, B Bockelman, J M Hernandez, E Vaandering

1 University of Notre Dame, Notre Dame, IN 46556, USA
2 Fermi National Accelerator Laboratory, Batavia, IL, USA
3 INFN Sezione di Trieste, 34127 Trieste, Italy
4 University of Nebraska-Lincoln, USA
5 CIEMAT, Madrid, Spain

E-mail: {mwolf3|awoodard}@nd.edu, marco.mascheroni@cern.ch

Abstract.

CRAB3 is a workload management tool used by CMS physicists to analyze data acquired by the Compact Muon Solenoid (CMS) detector at the CERN Large Hadron Collider (LHC). Research in high energy physics often requires the analysis of large collections of files, referred to as datasets. The task is divided into jobs that are distributed among a large collection of worker nodes throughout the Worldwide LHC Computing Grid (WLCG).

Splitting a large analysis task into optimally sized jobs is critical to efficient use of distributed computing resources. Jobs that are too big will have excessive runtimes and will not distribute the work across all of the available nodes. However, splitting the project into a large number of very small jobs is also inefficient, as each job creates additional overhead which increases load on infrastructure resources.

Currently this splitting is done manually, using parameters provided by the user. However the resources needed for each job are difficult to predict because of frequent variations in the performance of the user code and the content of the input dataset. As a result, dividing a task into jobs by hand is difficult and often suboptimal.

In this work we present a new feature called “automatic splitting” which removes the need for users to manually specify job splitting parameters. We discuss how HTCondor DAGMan can be used to build dynamic Directed Acyclic Graphs (DAGs) to optimize the performance of large CMS analysis jobs on the Grid.

We use DAGMan to dynamically generate interconnected DAGs that estimate the processing time the user code will require to analyze each event. This is used to calculate an estimate of the total processing time per job, and a set of analysis jobs are run using this estimate as a specified time limit. Some jobs may not finish within the allotted time; they are terminated at the time limit, and the unfinished data is regrouped into smaller jobs and resubmitted.

1. Introduction

Within CMS, user analysis is generally performed using CRAB3 [5], a workload management tool that provides a user-friendly interface to handle the processing of large quantities of files in a high-throughput environment on the Worldwide LHC Computing Grid (WLCG) [1]. CRAB3 provides a portal for users to submit projects to the WLCG. Users submit tasks to a central CRAB3 server, which then distributes the work to connected sites. Each task processes one
Figure 1. Workflow for a user submission of a task to the CRAB3 infrastructure. Tasks are accepted by a frontend and processed in several steps. First, metadata about the dataset the user wants to run over is acquired, which is used to split the task into jobs. Using this partitioning, a DAG is created to describe the job execution, and submitted to a HTCondor scheduler, where DAGMan will execute pre-, job, and post-jobs as specified in the DAG. Also shown, dashed, is the dry-run splitting estimation, which runs on the user machine and resumes the workflow with the submission to the scheduler after the splitting has been confirmed by the user.

collection of files, called a dataset. The datasets are located at various sites distributed across the WLCG. By default CRAB3 assigns processing to be completed at a location where the requisite input dataset is already stored, to minimize wide area network load and latency.

The CRAB3 system is split into several components for better modularity and scaling. The frontend provides a user-friendly interface, accepts user tasks, and performs some validity checks on the configuration of each task. It stores the task metadata in a database, from which the CRAB3 server process retrieves it in preparation for submission to the WLCG.

The CRAB3 server then acquires metadata about the dataset to be processed from the CMS Dataset Bookkeeping system (DBS [3]). According to the user’s task configuration, it then creates a specification for executing these jobs in the form of Directed Acyclic Graph (DAG).

The DAG is then transferred to a scheduler running HTCondor, a software framework for coarse-grained distributed parallelization of computationally intensive jobs, and executed by the DAG manager (DAGMan) [2], a feature of HTCondor.

DAGMan is a high-level scheduler. It interprets and executes the DAG, using it to analyze the order in which jobs must be executed, and generates instructions to HTCondor, the low-level scheduler. For every job in the DAG, DAGMan will execute a pre-job command on the
scheduler, which performs some preparation for the following job execution.

The job is subsequently submitted to HTCondor by DAGMan, which assigns it to a site in the WLCG for execution. Upon completion of each job, DAGMan initiates a post-processing command on the scheduler, which determines whether the job exited successfully or failed. If the job ran successfully, the output is transferred to the final storage element by the AsyncStageOut (ASO) server [6]. If the job failed, DAGMan flags it for re-submission. A graphical depiction of the process can be found in Figure 1.

DAGMan also makes it possible to execute multilevel DAGs, in which jobs can be marked as parents with one or more dependent (child) jobs, each with their own pre- and post-job scripts. The entire DAG can be submitted in a single file, or dependent parts of the graph can be submitted as separate files, to be parsed and executed only if the parent job finishes successfully.

The current CRAB3 structure has been widely accepted across the CMS user community and usage has increased considerably. As a result scaling issues have become apparent. For example, it has become necessary to limit the number of DAGs and post-job processes running concurrently on each scheduler to avoid failures due to excessive load.

2. Job Splitting

CRAB3 imposes a limit on the total number of jobs that each task may contain, but otherwise the task splitting is performed according to user preferences. As the splitting parameters must be supplied by the user, they are by necessity estimates based on prior experience running the job executable locally, or based on the results of a previous task submission. Such estimates are difficult given the frequent changes in both the software environment and the datasets being examined.

But as the software environment and its configuration, as well as the computing environment, change over the course of subsequent task submissions, splitting parameters have to be frequently changed to maintain similar job runtimes. We have noticed, as shown in Figure 2, that most user jobs have run times that are too short for optimal processing. A large number of short
Creating temporary directory for dry run sandbox in /tmp/tmpZiyjC_
Executing test, please wait...

Using LumiBased splitting  
Task consists of 16 jobs to process 30881 lumis  
The longest job will process 2000 lumis, with an estimated processing time of 12874 minutes  
The average job will process 1930 lumis, with an estimated processing time of 10861 minutes  
The shortest job will process 1191 lumis, with an estimated processing time of 6023 minutes  
The estimated memory requirement is 662 MB

Timing quantities given below are ESTIMATES. Keep in mind that external factors such as transient file-access delays can reduce estimate reliability.

For ~480 minute jobs, use:  
Data.unitsPerJob = 85  
You will need to submit a new task

Dry run requested: task paused  
To continue processing, use 'crab proceed'

Figure 3. Output of the crab submit --dryrun command. Following the estimation of the runtime, the user can confirm the current set of splitting parameters to continue processing, or resubmit the task with updated parameters.

jobs can cause excessive load on the submission infrastructure, and to prevent this it has become necessary to impose a minimum time before the next job can be scheduled, even if the job is completed. A better approach would be to split user tasks into a smaller number of somewhat larger jobs. However if the jobs are too large other inefficiencies appear, as the work is not efficiently distributed across the available resources. To mitigate these problems we have begun to provide capabilities to assist the user in task splitting, with the goal of ultimately automating the entire process.

2.1. Splitting by Files, Lumisections, or Events

CRAB3 provides a basic form of task splitting by specifying the number of files every job of a task should process, however individual files may of course contain different amounts of data. Users can refine the splitting process by instead assigning each job a specified number of lumisections ¹. However, the number of events in a lumisection may itself not be uniform within a file or across different sets of input data, so splitting based only on number of lumisections may also produce jobs that vary considerably in size. Consequently a third splitting mode was developed, event-aware lumi-based splitting, in which the number of lumisections assigned to each job is adjusted based on the number of events it contains, so that each job contains approximately the same number of events.

2.2. Dry-run Estimation of CPU Time and Memory Requirements

We implemented an additional improvement of the splitting process in the form of a dry-run command. This command stops the submission process on the CRAB3 server after the task has been split but before the generated DAG is submitted to the scheduler. Dry-run then downloads the files needed to run a standard job from the CRAB3 infrastructure to the local user machine and executes the job iteratively until a reliable estimate of the CPU load per event has been obtained. The user is then given an overview with the estimated job runtime, as well as a recommendation for improved splitting parameters. The user can proceed with the original task

¹ Sub-section of a run during which time the collision instantaneous luminosity is unchanging (roughly 23 seconds). Minimum granularity to analyze event data.
Figure 4. Splitting process for the automatic splitting mode: A short probe job is run first in the main DAG. In the corresponding postjob process, the splitting and DAG creating components of the CRAB3 server code are used to fill a SubDAG with instructions to run the processing jobs with optimized splitting.

This approach requires the user to submit each task to the CRAB3 server, then tune the parameters and submit a new task if the estimated runtime is not correct. This improved performance but proved cumbersome for the user at times, particularly when multiple jobs are being submitted through a script or client process. This led us to introduce the new Automatic Splitting mode.

2.3. Automatic Splitting

We felt ease of use and Grid resource usage could be optimized only if we could eliminate the need for the user to manage task splitting entirely, and instead use the CRAB3 submission infrastructure to perform this task autonomously. The Automatic splitting mode allows the user to directly specify the desired job runtime, with no need for further interaction.

The estimation of processing time per event cannot be performed on the CRAB3 server because it does not have the capacity to run estimation jobs for all the users, and performing a dry run on the user’s machine is not an option because we cannot control it remotely. The solution we implemented is to let DAGman run an estimating job, then perform the splitting operation and run the full task.

The CRAB3 server prepares a DAG that includes a small probe job, but also has an empty subDAG. We use a new feature in the CMS software (CMSSW) suite that makes it possible to set a limit on runtime. The probe job runs for a specified period on a single node and then exits, allowing us to measure the CPU time and memory requirements. When the probe job reaches the time limit it terminates gracefully, returning the results it has generated.

In the post-processing segment of the probe job we run the splitting procedure using the estimated time per event computed in the probe, and then we fill the subDAG with the list of processing jobs resulted from the splitting. The use of the subDAG feature allows CRAB3 to generate and submit the DAG even though the splitting parameters are unknown at the time.
of submission.

2.4. Tail jobs
A third improvement for automatic task splitting is to address jobs that, despite the estimation process, cannot finish by the predicted time. There are numerous causes for this, including variations in the hardware configurations of the worker nodes, variations in data transfer rates due to loading of of the site storage elements, variations in load on the wide area network if remote data access is performed with XrootD [4], variations in event complexity, and variations in the number of events per luminsection.

Allowing a few tail jobs to run indefinitely is suboptimal because it does not spread the work across available resources efficiently. Moreover, to stop runaway jobs there are hard limits on total run time, and if a job reaches such a limit it is terminated and the results are lost. In order to limit runtime without data loss we use the CMSSW runtime limit here as well. With the data from the probe job, a run time limit is calculated for the processing jobs.

If a processing job exceeds this limit, it is terminated and the post-job script returns the completed results and reports the uncompleted input. The splitting algorithm is run on this input and smaller tail jobs are resubmitted that utilize more of the available worker nodes and minimize the time to completion of the full task, as shown in Figure 5.

3. Conclusion
In this paper we describe how the new “automatic splitting” feature uses HTCCondor DAGMan to remove the need for users to manually specify job splitting parameters.

DAGMan has been used to dynamically generate interconnected DAGs that estimate the processing time the user code will require to analyze each event, create a set of analysis jobs that perform the bulk of the analysis, and finally send tail jobs to analyze the missing data from jobs that did not finish within the allotted time. This is made possible by the subDAG feature that allows changes to be made to the DAG dynamically, during the execution process.

This provides users with assurance that tasks will be completed in a shorter and more predictable time interval, and ensures that the system resources are used more efficiently by optimizing task splitting.
Challenges for the future include improving the re-splitting of tail jobs and otherwise continuing to automate the analysis process, to allow users to focus on their research rather than on the internal details of the workload management system.

References