FERMILAB-CONF-08-556-CD-CMS

International Conference on Computing in High Energy and Nuclear Physics (CHEP'07) IOP Publishing
Journal of Physics: Conference Series **119** (2008) 062033 doi:10.1088/1742-6596/119/6/062033

WLCG scale testing during CMS data challenges

Oliver Gutsche[1], **Csaba Hajdu**[2]

[1] Fermi National Accelerator Laboratory, CMS/CD MS 205, P.O.Box 500, Batavia, IL, 60510, USA

[2] Research Institute for Particle and Nuclear Physics, P.O.Box 49, Budapest, H-1525 Hungary

E-mail: gutsche@fnal.gov

E-mail: hajdu@sunserv.kfki.hu

Abstract. The CMS computing model to process and analyze LHC collision data follows a data-location driven approach and is using the WLCG infrastructure to provide access to GRID resources. As a preparation for data taking, CMS tests its computing model during dedicated data challenges. An important part of the challenges is the test of the user analysis which poses a special challenge for the infrastructure with its random distributed access patterns. The CMS Remote Analysis Builder (CRAB) handles all interactions with the WLCG infrastructure transparently for the user. During the 2006 challenge, CMS set its goal to test the infrastructure at a scale of 50,000 user jobs per day using CRAB. Both direct submissions by individual users and automated submissions by robots were used to achieve this goal. A report will be given about the outcome of the user analysis part of the challenge using both the EGEE and OSG parts of the WLCG. In particular, the difference in submission between both GRID middlewares (resource broker vs. direct submission) will be discussed. In the end, an outlook for the 2007 data challenge is given.

1. Introduction

The Large Hadron Collider (LHC) at CERN, Geneva, Switzerland [1] is nearing its completion with first proton-proton collisions expected for July 2008. CMS, the Compact Muon Solenoid [2], one of the two startup experiments of the LHC, underwent a still ongoing extensive test program of the detector and the data handling systems to be ready to record data beginning at the start of collisions in the LHC.

A part of these test is the CMS computing model [3] which deals in general with the handling of recorded data and simulated events. It is based on a worldwide distributed tier structure of regional and local data centers which store and provide distributed access to the complete data of the CMS experiment. The tier structure uses GRID tools to manage all handling of and access to the distributed data samples.

The access to data by the ~2000 members of the CMS collaboration plays an important role within the CMS computing model. Physicists will access recorded and simulated events and analyze them to extract physics results to possibly discover the Higgs and/or new physics beyond the standard model.

To facilitate the access to the samples for the user, CMS developed a user tool called the **C**MS **R**emote **A**nalysis **B**uilder (*CRAB*) [6] which hides interactions with the GRID middlewares and provides the user with an unified and user-friendly interface.

© 2008 IOP Publishing Ltd

Part of the test program of the CMS computing model are dedicated data challenges which test components of the model at a defined scale. This paper describes the experiences and results of the data challenge **C**omputing, **S**imulation and **A**nalysis 20**06** (*CSA06*) [10] with emphasis on the test of the user analysis infrastructure using CRAB.

After the introduction of the CMS computing model, the user analysis workflow and the user analysis GRID tool CRAB, the user analysis part of CSA06 and the used test infrastructure are described. In the end, the user part of the challenge will be summarized and an outlook of the upcoming data challenge CSA07 in 2007 will be given.

2. CMS computing model

The CMS computing model deals in general with the handling of recorded data of the CMS detector and simulated CMS events. CMS chose early on to distribute its needed resources worldwide to provide sufficient computing power for the experiment due to many reasons (e.g. funding, infrastructure).

The distributed computing structure of CMS consists of three main levels or tiers. The **tier 0** (*T0*) is located at CERN where the accelerator and experiment are located and represents 20% of the total required computing resources of CMS. The next level is represented by 7 **tier 1** (*T1*) regional centers which represent 40% followed by the 25-50 **tier 2** (*T2*) centers which represent another 40% of the total required computing resources of CMS (see Fig. 1).

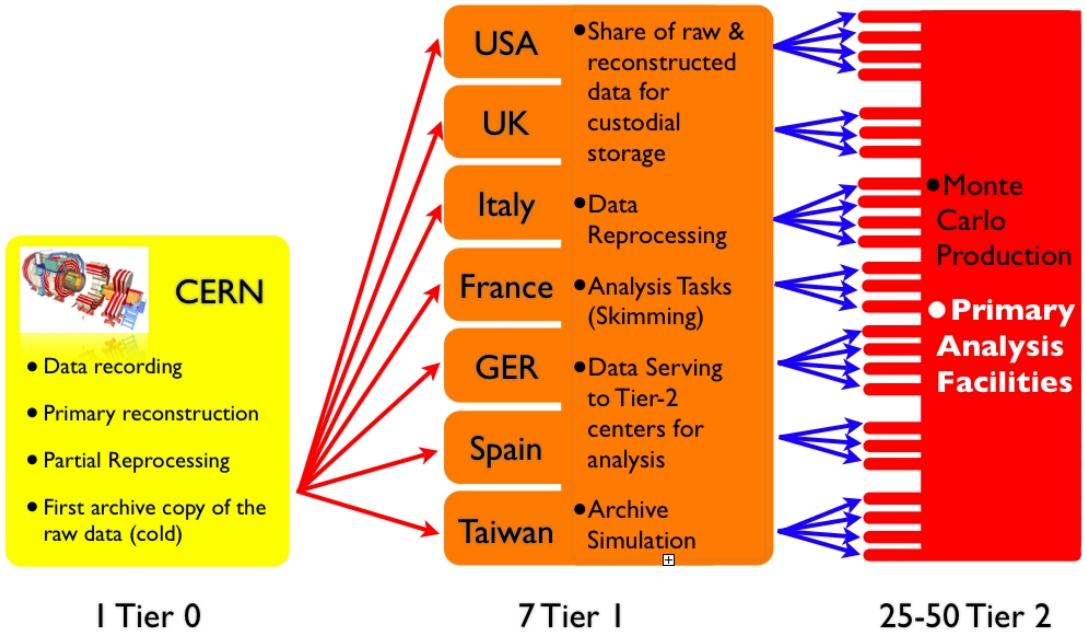

Figure 1. Overview of the CMS computing model. Shown are the 3 main tier levels and their respective responsibilities concerning data handling.

The responsibilities of the tier levels are differentiated by their resource dedication and type of data storage:

Tier 0:

- Data is recorded by the detector.
- Raw data is archived on tape as an inaccessible copy (cold copy).
- Prompt reconstruction of the recorded data is carried out including splitting in primary datasets.
- Primary datasets are distributed amongst the T1 centers.

Tier 1:

- Raw and reconstructed data from the T0 is shared for custodial storage on tape amongst the T1 centers.
- Simulated events are archived on tape.
- Central processing operations like re-reprocessing, skimming and extraction of the **A**nalysis **O**bject **D**ata event format (*AOD*, main and smallest event format for physics analysis) are carried out.
- Data samples in various formats are distributed to the T2 level.

Tier 2:

- No tape systems are available at T2 centers.
- Data from T1 centers is cached on disk and provided for analysis.
- The T2 level represents the primary analysis facilities for CMS.
- Simulation of Monte Carlo events is carried out on the resources of the T2 level and archived on tape back on the T1 level.

Users access data samples and Monte Carlo events primarily on the T2 level using cached samples from the T1 level. CMS follows a GRID approach to provide access to this samples using the WLCG middlewares. The access is location driven, central processing and analysis jobs are sent to a center which stores or caches the requested data. The access is handled by the user analysis GRID tool CRAB described in the next section.

3. CMS Remote Analysis Builder (CRAB)

CMS provides a user tool to hide all interaction with the various GRID systems called the **C**MS **R**emote **A**nalysis **B**uilder (*CRAB*) [6]. CRAB's purpose is to execute the user's analysis code on remote resources as if the user would execute it interactively. CRAB has to

(i) Collect the user's code and prepare it for shipping to the workernode,

(ii) Discover the location of the input dataset requested by the user and split the project into jobs according to the user's specification,

(iii) Submit the jobs to the remote resource and run the user's analysis code on the requested input dataset,

(iv) Provide status information and collect the user's output and transport it back to the user.

CRAB uses various CMS and WLCG systems to carry out the user analysis workflow shown in Fig. 2. In addition to the EGEE [4] and OSG [5] GRID middlewares to access T1 and T2 resources, CRAB uses CMS' **D**ataset **B**ookkeeping **S**ystem (*DBS*) [7] and CMS' **D**ata **L**ocation **S**ervice (*DLS*) [8] to discover availability and location of datasets within the CMS tier structure.

CRAB groups all steps into four easy-to-use user commands:

crab -create collects the user's code, discovers the dataset and performs job splitting

crab -submit submits the jobs using the chosen GRID middleware

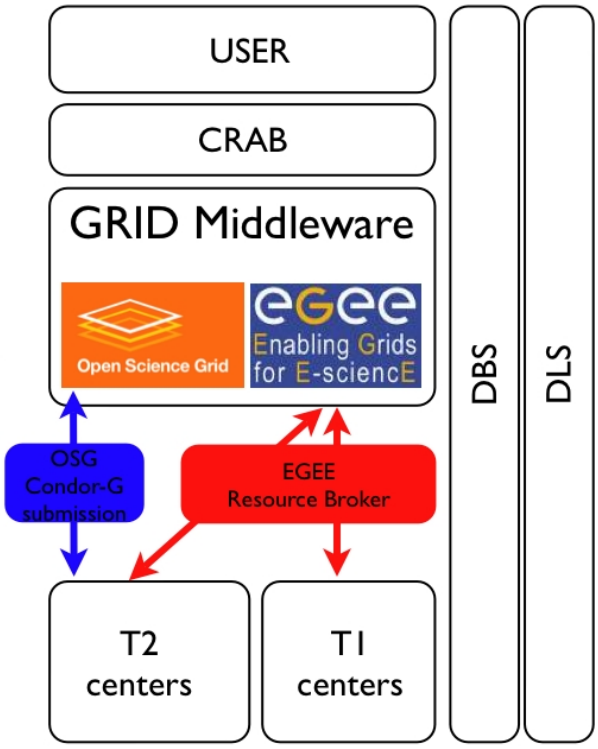

Figure 2. Analysis workflow implemented by CRAB from the user project definition to the execution on the workernode. Amongst the EGEE and OSG middlewares, CRAB uses DBS and DLS to discover the availability and location of datasets within the CMS tier structure.

crab -status gathers status information about the submitted jobs

crab -getoutput retrieves the user output

Underneath, CRAB uses the *CMS Bookkeeping System* (BOSS) [9] to manage all jobs locally on the user's machine. BOSS builds on a SQLite or MySQL database to keep track of jobs and their status.

In the following, the data challenge CSA06 is presented with emphasis of the test of the user analysis infrastructure at high scales.

4. Computing, Software and Analysis challenge 2006 (CSA06)

In 2006, CMS conducted a data challenge called **C**omputing, **S**imulation and **A**nalysis 20**06** (*CSA06*) [10] to test CMS' computing infrastructure, data flow and data handling at a scale of 25% of the requirement for CMS operations in 2008. The tests consisted of almost all components of the CMS computing model:

- Prompt reconstruction at T0 including dataset distribution to T1 level,
- Calibration, re-reconstruction and skimming at T1 level and dataset distribution to T2 level,
- High scale user analysis job tests using CRAB on T2 level.

The goal of the high scale tests of the job submission infrastructure was 50,000 submitted jobs per day where about 10,000 jobs per day were central production jobs on the T1 level like

re-reconstruction and skimming and about 40,000 jobs per day CRAB analysis jobs. The latter should be reached by supplementing job submissions from the user base with automated user analysis jobs submitted by JobRobots.

The challenge was held from 02. October to 15. November 2006. During the challenge, 948,099 jobs were submitted in total. The number of jobs of the different categories (production, analysis and JobRobot) can be seen in Fig. 3.

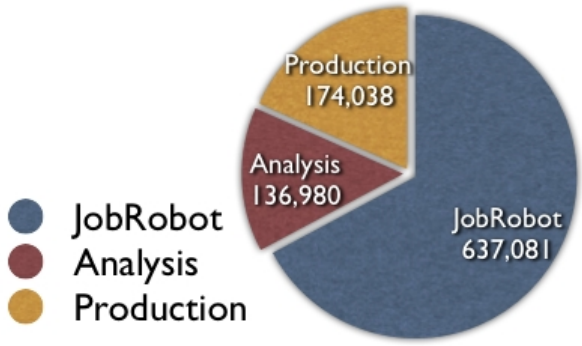

Figure 3. Total number of submitted jobs during CSA06 (948,099) separately shown by category: production, analysis and JobRobot.

The largest fraction of the total submitted jobs was submitted by the automated user analysis submission infrastructure which is described in the next section.

4.1. Automated user analysis infrastructure: JobRobot

The purpose of the automated test infrastructure called the *JobRobot* is to run analysis-like jobs using CRAB to supplement standard user activity to reach high scales during challenges.

The JobRobot is implemented using a perl agent framework. The different agents of the JobRobot follow the different steps of the user analysis workflow and are arranged in levels following the CRAB usage steps of creation, submission, status check and output retrieval.

The Perl agent framework allows to operate several agents of the same type in parallel operating on individual tasks passed between the different agent levels using input and output queues. A task is a specific CRAB project to perform an analysis on a defined dataset at a specified site.

The JobRobot framework submits continuously simple CRAB jobs using the CMS framework CMSSW [11] to read in and write out events of the specified input dataset. A typical JobRobot job runs for half an hour. The output is discarded on the workernode, only the log-file is transported back to the JobRobot. The JobRobot is designed to submit jobs to all datasets known to DBS and to all sites hosting those datasets while sustaining a constant job rate at individual sites to avoid burst submission.

In the following, the different agents and their functionality are described in their order of execution to simulate the user analysis workflow:

TaskSource: The *TaskSource* performs a complete discovery of all CMS datasets and all their locations to prepare a CRAB configuration per dataset/site combination representing a JobRobot task. It simulates the part of the user workflow which is usually performed outside of CRAB, the decision of the user on which dataset to run her/his analysis. It restricts the creation of new tasks to sustain a constant job load at all sites and avoid burst submissions. One single instance of the TaskSource is creating tasks and sends them to the input queue of the next level.

TaskPrepare: Several *TaskPrepare* agents execute the creation step of CRAB on input tasks. Here the previously discovered dataset/site combination is discovered again by the inbuilt CRAB discovery and the individual jobs of the task are created using the defined job splitting. The created tasks are handed over to the input queue of the next level.

TaskSubmit: Several *TaskSubmit* agents take input tasks and execute the submission step of CRAB. Also here, the job submission to individual sites is restricted to sustain a constant load at the sites. The successfully submitted tasks are send to the input queue of the next level.

TaskQuery: Several *TaskQuery* agents repeatedly check the status of all tasks performing the status check part of the CRAB user analysis workflow. The *TaskQuery* is the most resource intensive step of the JobRobot structure. It cleans up aborted jobs and resubmits them for a defined number of times. The agent sends tasks to the input queue of the next level if all jobs of the task are done or finally aborted.

TaskCollect: Several *TaskCollect* agents use CRAB to retrieve the log-files of the jobs of a task. A post mortem analysis is performed and logging information is collected. After all jobs of a task have been treated by this agent, the task is closed and removed from the queues.

The JobRobot agents contain sufficient error handling procedures to guarantee a continuous and unattended operation. The experiences with the JobRobot infrastructure and the efforts to reach the goal of 50,000 jobs a day for CSA06 are described in the next section.

4.2. JobRobot operation during CSA06

Automated submission of analysis-like CRAB jobs started 15. October 2006 at a rate of 10,000 jobs per day. In the following, the time-line of JobRobot operation shown in Fig. 4 is discussed in more detail.

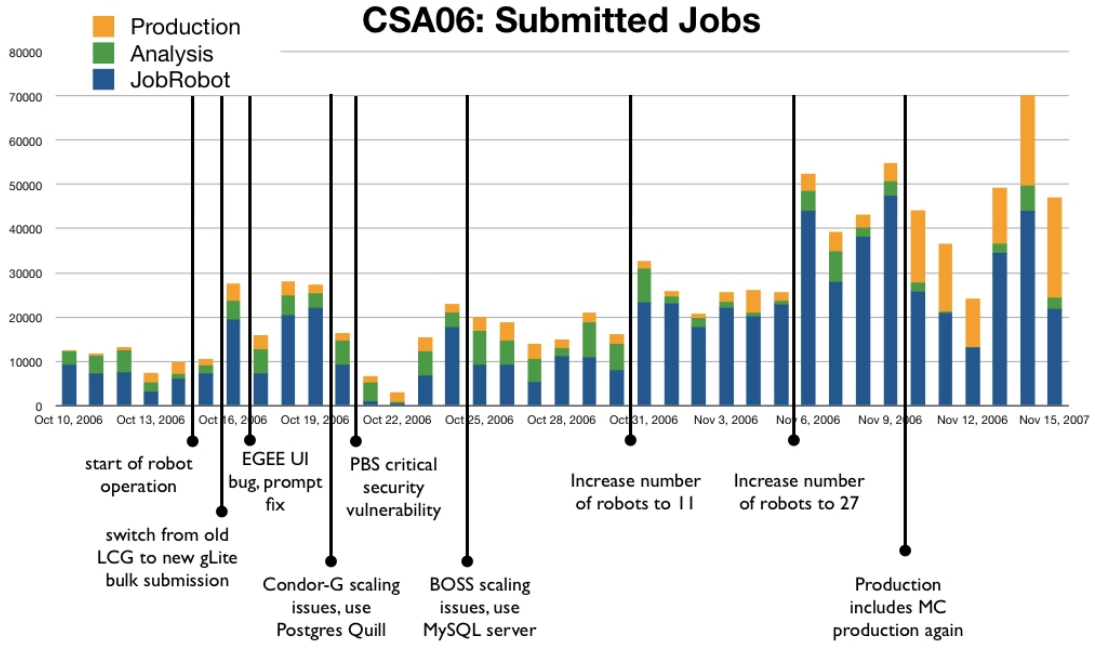

Figure 4. Time-line of JobRobot operation during CSA06.

15. October 2006: Startup: At the startup of JobRobot operation on 15. October 2006, two machines were used initially to run two instances of the JobRobot, one using the direct submission mode using Condor-G [12] to OSG T2 sites and one using the EDG resource broker [13] to submit to LCG sites. As can be seen in Fig. 5, the EDG resource broker did not provide sufficient performance compared to the OSG direct submission using Condor-G.

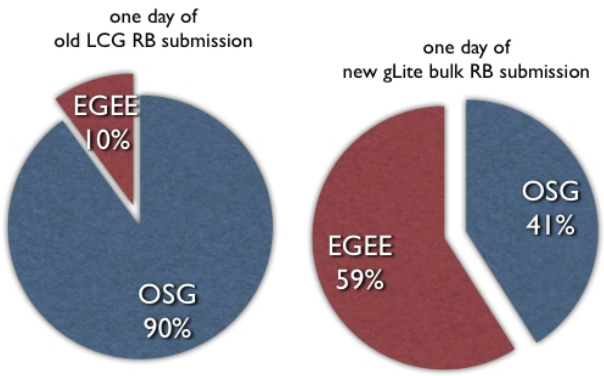

Figure 5. Compared are the fraction submitted to EGEE and OSG per day using the LCG resource broker *(left)* and the newly in CRAB integrated gLite bulk submission *(right)*.

After integrating the much higher performant gLite bulk submission mode [14] into CRAB, the lower performance of the LCG JobRobot was compensated and the job submission rate more than doubled to almost 20,000 jobs per day. During the transition to use the gLite bulk submission, a bug was found in the gLite user interface and promptly fixed by the gLite developer team.

19. October 2006: Condor scaling issues: The OSG JobRobot developed scaling issues starting 19. October 2006. The continuous status checks of the TaskQuery agents overloaded the local Condor scheduler and increased the response time of checking the queue significantly and so reducing the job submission rate. This was improved by installing a Quill Postgres database [12] for the local Condor scheduler which inserts a database between the user and the local Condor queue. In addition, several OSG sites did not accept new jobs due to security issues with their local PBS batch systems. Both problems had been recovered on October 24. 2006 and the submission rate was back to more than 20,000 jobs per day.

24. October 2006: SQLite scaling issues: On 24. October 2006, both robots showed again reduced performance. The reduction was traced back to an significantly increased I/O rate on the local disks which originated from the large number of SQLite interactions of CRAB while performing mainly the status check step. Too many jobs were submitted per site as status checks could not be performed in a timely manner to restrict the submissions. CE's of several sites were killed during this period and had to be rebooted. As a solution, a central MySQL database server for CRAB/BOSS was setup and used by both JobRobot instances.

24. October 2006: gLite job submission efficiency issues: Further reduced performance of the EGEE JobRobot was traced back to very low submission efficiencies of tasks with more than 2000 jobs. To recover the lost efficiency, the number of jobs per tasks were restricted to be less than 200.

24. October 2004: BOSS SQL query fix: A timeout problem of the EGEE JobRobot was observed which decreased performance. After installing a second MySQL database server

for CRAB/BOSS and having both JobRobot instances use their own MySQL server, the problem of the EGEE JobRobot was traced back to an incomplete MySQL query and fixed by the BOSS developer team.

31. **October 2006: Increased number of JobRobot instances:** Till October 31, all tuning and improvement measures could not increase the scale to constantly more than 10,000 jobs per day. Concurrent operations of all agents slowed down the JobRobots due to many I/O accesses to various log-files and frequent database queries to MySQL and postgres. By increasing the number of individual JobRobot instances on separate machines to 11 and going back to use SQLite for CRAB/BOSS and the normal Condor scheduler without a Quill Postgres database, the performance could be increased to 20,000-25,000 jobs per day.
5. **November 2006: Further increase in number of JobRobot instances:** A further increase of the performance was reached on 5. November 2006 by increasing the number of JobRobot instances to 27.
6. **November 2006: CSA06 goal reached:** On November 6. 2006, the goal of 50,000 jobs per day was reached (see Fig. 6). 57% of the jobs were run on OSG resources while 45% on EGEE resources which corresponded to the approximate resource distribution between sites on EGEE and OSG.

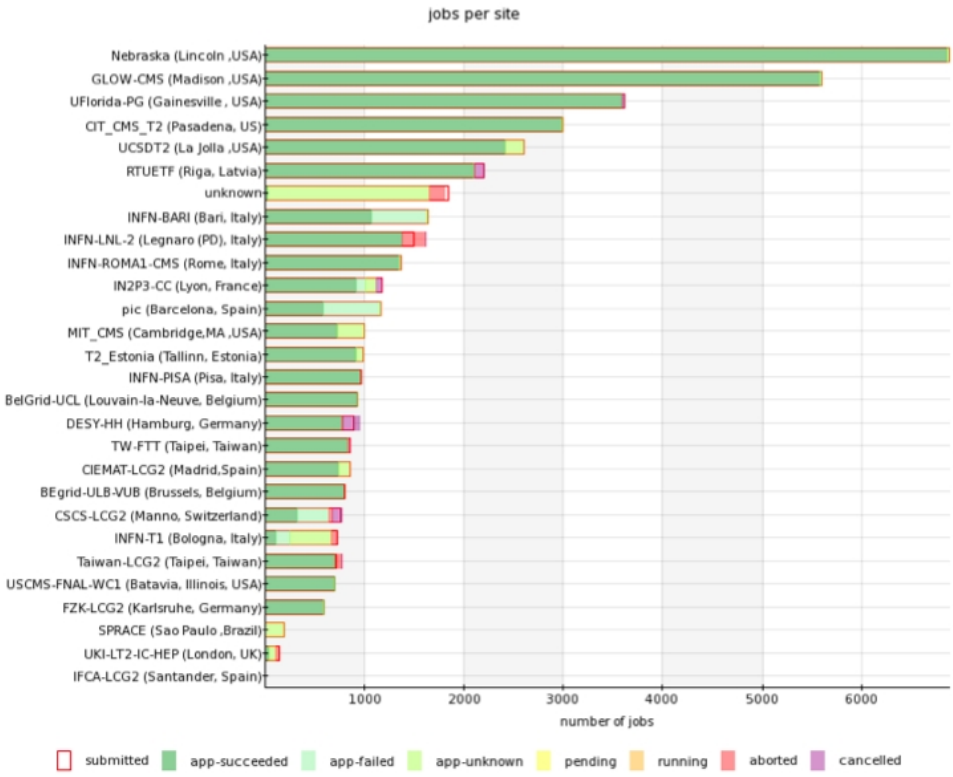

Figure 6. On November 6. 2006, the goal of 50,000 jobs per day was reached. Shown is the distribution of JobRobot jobs per site.

A scale of 40,000 jobs per day could be maintained for several days and a record of more than 12,000 jobs per day submitted to a single site (Florida) was reached on 10. November 2006 when also MC production re-started again on the T1 and T2 resources.

In the end, the required scale of 50,000 jobs per day was only reached by increasing the number of JobRobot instances and therefore reducing the number of destination sites per instance significantly. A combination of reasons led to the necessity of increasing the number of JobRobots: first the robotic execution of the stand-alone CRAB user tool which introduced significant load on the hardware if many CRAB operations were executed in parallel; second the frequency of status checks necessary to track the complete job status of a robot which again stressed the local hardware using the GRID middlewares significantly.

Solutions are already available or become available. The first problem is already solved in current production systems [15] which integrate JobRobot functionalities and job submission tools efficiently. The second will be solved with the introduction of the server based CRAB service [16] which will more efficiently query the GRID middleware and provide status information in a central way.

On the other hand, this will be not an issue for real user interaction as many users will submit jobs from many systems and don't have to keep a constant load at the sites so making less frequent status queries.

5. Job submission efficiency

Summarizing the JobRobot operation in terms of submission efficiency, four different submission status results can be distinguished:

Success: GRID submission including status check and output retrieval was successful.

Canceled: GRID job was submitted but canceled by the JobRobot due to problems or too long running times (internal timeout to avoid runaway jobs).

Unknown: Problems with the monitoring where no or incomplete monitoring information reached the central collector or problems with the GRID infrastructure where the job never reached the workernode and could not return its running location are tracked under the *unknown* status.

Aborted: GRID jobs which have been aborted by the GRID infrastructure are counted as *aborted*.

The summary of the submission efficiency shown in Fig. 7 shows an overall efficiency of 71% with a very low percentage of *canceled* or *aborted* jobs. The fraction of *unknown* jobs currently cannot be separated in GRID failures and monitoring problems.

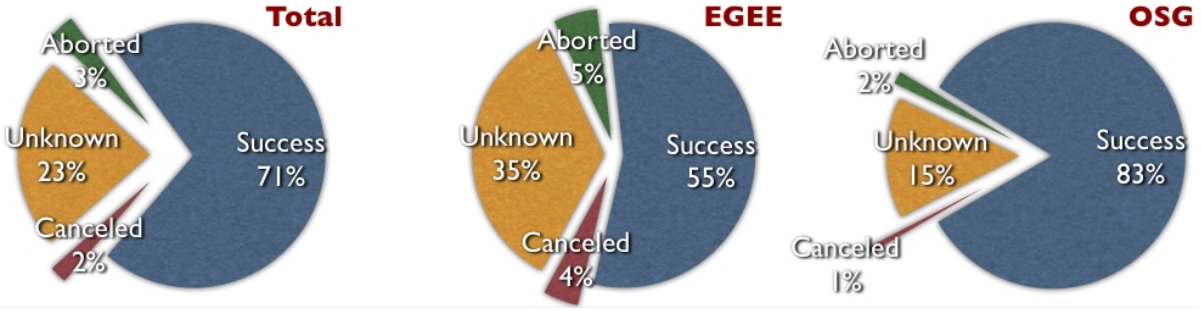

Figure 7. Job submission efficiency during CSA06. Shown is the overall efficiency *(left)* and the respective efficiencies for EGEE *(center)* and OSG *(right)*.

6. Summary and Outlook

The goal of 50,000 job submissions per day during CSA06 was only reached by increasing the number of automated job submission instances (JobRobots) significantly from the initial two to 27 instances. The requirement of constant load at individual sites required frequent status checks of all jobs of an individual JobRobot which put a large load on the individual system due to several bookkeeping interaction with the local and GRID infrastructure. This load was reduced to an acceptable level by reducing the number of sites served by a JobRobot therefore increasing the number of JobRobot instances. In the end, the overall submission efficiency of 71% was acceptable and the analysis goal for CSA06 was reached.

For 2007, CMS plans the next data challenge at 50% scale of what is needed for operation in 2008. The analysis part will have to reach over 100,000 jobs per day. Compared to CSA06, CMS will primarily rely on the user base to submitted all the analysis jobs. This should be possible due to an recently increased user activity using Monte Carlo samples and test-beam and test-stand data. This will be the next challenging test of the user analysis workflow. It will not be dominated by controlled job submissions to keep steady loads at sites but rather process random user job submissions which will test the infrastructure in a much more realistic way.

7. Acknowledgments

We would like to thank the funding agencies under which this work was conducted. The presented tests used infrastructures from several GRID middlewares and tools and services developed and provided by CMS who we want to thank for their help, support and contributions.

References

[1] O. Bruning, P. Collier, P. Lebrun, S. Myers, R. Ostojic, J. Poole and P. Proudlock, "LHC design report. Vol. I: The LHC main ring", *CERN-2004-003*
[2] CMS Collaboration, "CMS, the Compact Muon Solenoid: Technical proposal",*CERN-LHCC-94-38*
[3] CMS Collaboration, "CMS: The computing project. Tech- nical design report", *CERN-LHCC-2005-023*
[4] Enabling Grids for E-sciencE http://www.eu-egee.org/
[5] Open Science Grid http://www.opensciencegrid.org/
[6] F. Fanzago et al, "CRAB: a tool to enable CMS Distributed Analysis", Proceedings for Computing in High-Energy Physics (CHEP '06), Mumbai, India, 13 Feb - 17 Feb 2006
[7] Lee Lueking et al, "The CMS Dataset Bookkeeping Service", Proceedings for Computing in High-Energy Physics (CHEP '07), Victoria B.C., Canada, 3 Sep - 7 Sep 2007
[8] A. Fanfani, "Distributed data management in CMS", Proceedings for Computing in High-Energy Physics (CHEP '06), Mumbai, India, 13 Feb - 17 Feb 2006
[9] W. Bacchi et al, "Evolution of BOSS, a tool for job submission and tracking", Proceedings for Computing in High-Energy Physics (CHEP '06), Mumbai, India, 13 Feb - 17 Feb 2006
[10] I. Fisk, "CMS Experiences with Computing Software and Analysis Challenges", Proceedings for Computing in High-Energy Physics (CHEP '07), Victoria B.C., Canada, 3 Sep - 7 Sep 2007
[11] C. Jones et al, "The new CMS event data model and framework", Proceedings for Computing in High-Energy Physics (CHEP '06), Mumbai, India, 13 Feb - 17 Feb 2006
[12] Condor http://www.cs.wisc.edu/condor/
[13] European Data Grid http://eu-datagrid.web.cern.ch/eu-datagrid/
[14] gLite http://glite.web.cern.ch/glite/
[15] D. Evans, "CMS MC Production System Development & Design", Proceedings for Computing in High-Energy Physics (CHEP '07), Victoria B.C., Canada, 3 Sep - 7 Sep 2007
[16] D. Spiga, "CRAB (CMS Remote Anaysis Builder)", Proceedings for Computing in High-Energy Physics (CHEP '07), Victoria B.C., Canada, 3 Sep - 7 Sep 2007