CMS distributed data analysis with CRAB3

2015 J. Phys.: Conf. Ser. 664 062038

(http://iopscience.iop.org/1742-6596/664/6/062038)

CMS distributed data analysis with CRAB3

M Mascheroni[1], J Balcas[2], S Belforte[3], B P Bockelman[4], J M Hernández[5], D Ciangottini[6], P B Konstantinov[7], J M D Silva[8], M A B M Ali[9], A M Melo[10], H Riahi[11], A J Tanasijczuk[12], M N B Yusli[9], M Wolf[13], A E Woodard[13], E Vaandering[14] for the CMS Collaboration

[1] Universita e INFN, Sezione di Milano-Bicocca,
[2] Vilnius University,
[3] Universita e INFN, Sezione di Trieste,
[4] University of Nebraska-Lincoln,
[5] CIEMAT, Madrid,
[6] Universita e INFN, Sezione di Perugia,
[7] Bulgarian Academy of Sciences,
[8] UNESP - Universidade Estadual Paulista,
[9] University of Malaya,
[10] Vanderbilt University,
[11] CERN,
[12] University of California San Diego,
[13] University of Notre Dame ,
[14] FNAL.

E-mail: marco.mascheroni@cern.ch

Abstract. The CMS Remote Analysis Builder (CRAB) is a distributed workflow management tool which facilitates analysis tasks by isolating users from the technical details of the Grid infrastructure. Throughout LHC Run 1, CRAB has been successfully employed by an average of 350 distinct users each week executing about 200,000 jobs per day.

CRAB has been significantly upgraded in order to face the new challenges posed by LHC Run 2. Components of the new system include 1) a lightweight client, 2) a central primary server which communicates with the clients through a REST interface, 3) secondary servers which manage user analysis tasks and submit jobs to the CMS resource provisioning system, and 4) a central service to asynchronously move user data from temporary storage in the execution site to the desired storage location. The new system improves the robustness, scalability and sustainability of the service.

Here we provide an overview of the new system, operation, and user support, report on its current status, and identify lessons learned from the commissioning phase and production roll-out.

1. Introduction

The Compact Muon Solenoid (CMS) is one of two multipurpose experiments that collect and analyze data from proton-proton and heavy ion collisions at the Large Hadron Collider (LHC). The offline data processing activities carried out in CMS can be classified into two categories: *organized processing* of data, performed by a central operations team, and *data analysis*, performed by individual users. These two stages are complementary; "primary datasets" are produced centrally, then given to all the members of the collaboration to perform further analyses and report results.

A critical tool for user data analysis within the CMS collaboration is the CMS Remote Analysis Builder (CRAB), a software suite that allows physicists to transparently access the global computing resources of the Worldwide LHC Computing Grid (WLCG [1]). CRAB accepts a user's analysis code and a list of configuration parameters, and manages all of the operations that ultimately create *jobs* that are distributed to worker nodes around the world for execution.

CRAB2 was the official production release during LHC Run 1. Significant limitations in CRAB2 were identified based on the experience gained during that period. Output data was transferred directly from the Grid worker nodes in an unscheduled way. Many administrative functions had to be controlled directly by the user, including the resubmission of jobs interrupted by problems in the system. Most of the operations the tool needed to perform to create jobs, like contacting other CMS services and the Grid schedulers, were done on the user's submission node. This made analyses cumbersome to perform and difficult to debug, since problems on the user's client system were not directly accessible to administrators. It was determined that needed improvements in both internal operations and the user experience would require a major revision which has been implemented with CRAB3.

The paper provides an overview of the improvements introduced by CRAB3, and it is organized as follows: Section 2 introduces the distributed computing model implemented at CMS; Section 3 presents the architecture of CRAB3; Section 4 describes the commissioning phase and production roll-out of CRAB3; finally, Section 5 outlines the operational improvements introduced and new directions of development.

2. The CMS distributed computing model

CMS computing activities take place around the world at computing centers organized as a hierarchy of *Tiers*. A single Tier-0 center at CERN performs prompt reconstruction and provides archival storage on tape. Seven Tier-1 centers perform organized processing and data intensive tasks. Approximately 50 Tier-2 centers perform physics *user* data analysis and simulated *central* event production on a 50%/50% basis. Finally, more than 80 Tier-3 are focused primarily on user analysis.

CMS computing activities utilize a range of tools which function in concert to provide the full spectrum of services needed for analysis. The Data Management (DM) group includes the Data Bookkeeping System (DBS [2]), an event data catalog for both simulated and recorded data, the Physics Experiment Data Export (PhEDEx [3]) system, which manages global data transfers over the grid, and SiteDB [4], which tracks all the sites and resources available to the CMS collaboration and manages user authentication. The Workload Management (WM) group includes WMAgent [5], used for central production of datasets, and CRAB, the primary tool for user analysis. Both of these workflow management tools use glideinWMS [6] to access Grid resources in an easy way, and both use the CMS glideinWMS global pool deployed at CERN [7].

CRAB permits physicists to analyze CMS data without requiring extensive familiarity with the Grid infrastructure or the CMS computing model. User code is compiled locally and then sent to the Grid worker nodes where the CMS software framework (CMSSW [10])is provided through the CVMFS distributed file system[11]. CMSSW is the overall collection of tools that facilitates the development of reconstruction and analysis software, and provides services for simulation, calibration, alignment and reconstruction.

When the user designates an input dataset, CRAB performs a *data discovery* operation to locate the associated file metadata in the DBS. This information is then combined with the *splitting parameters* provided by the user to determine how the task will be split into jobs and to identify the data each job will require. To minimize file transfer overhead, a dataset is normally pre-staged at a Tier-2 site with PhEDEx so that jobs requiring it can be sent there for execution. However, a job can also access data as needed anywhere on the Grid using XrootD[9]; this approach is called the Any data Anytime Anywhere paradigm (AAA [8]). Users can also perform Monte Carlo simulations with CRAB, which requires no input dataset.

3. CRAB3 architecture

CRAB3 is a major reworking of the functionality provided by CRAB2, which introduces many improvements. DBS metadata lookup, job splitting, and scheduler interactions have been moved from the client to the server, facilitating administrator access for troubleshooting and reducing the possibility of user-specific failures. The REST architectural style allows the use of HTTP headers and caching mechanisms that improve load balancing and reduce the workload on other CMS computing tools including DBS, PhEDEx and SiteDB. Simultaneous requests from multiple users are now queued in the server, reducing the risk of overloading external services. Finally, an internal database for job submission and bookkeeping used in CRAB2 was replaced with the DAGMan (Directed Acyclic Graph Manager) meta-scheduler for HTCondor which manages dependencies between jobs at a higher level than the HTCondor Scheduler [?]. Where possible, code from the organized processing system is reused.

Figure 1 shows the complete architecture of CRAB3. The CRAB3-specific components are shown in green.

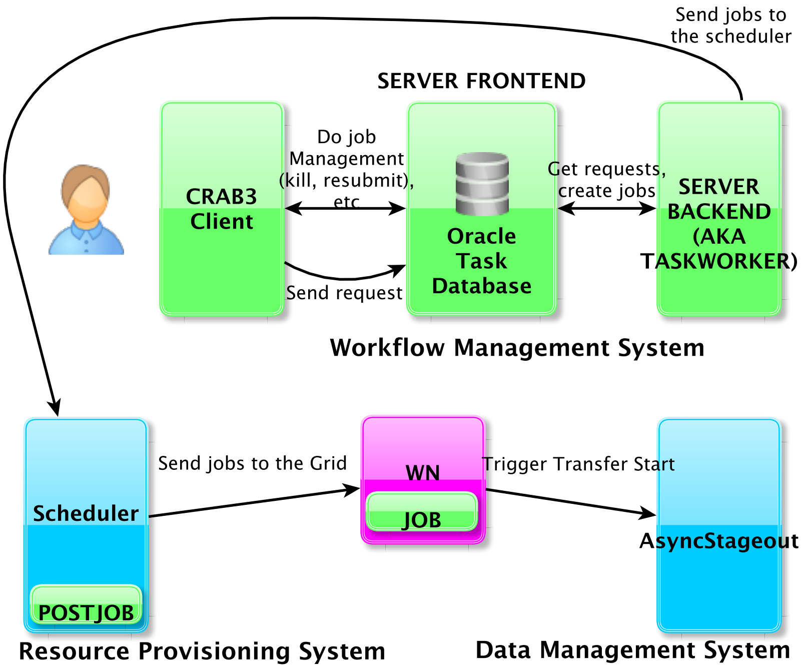
Send jobs to the scheduler
SERVER FRONTEND
CRAB3 Client
Do job Management (kill, resubmit), etc
Send request
Oracle Task Database
Get requests, create jobs
SERVER BACKEND (AKA TASKWORKER)
Workflow Management System
Scheduler
POSTJOB
Send jobs to the Grid
WN
JOB
Trigger Transfer Start
AsyncStageout
Resource Provisioning System
Data Management System

Figure 1. The overall architecture of the CRAB3 system.

The lightweight CRAB3 client is distributed through cvmfs and only requires *python* and *pycurl* to run; it provides access to all the CRAB3 client functions through a command line interface. Principle commands include status, submit, kill, resubmit, report, getoutput and getlog. The submit command loads the user configuration, does an initial validation, and sends it to the server frontend along with the user code. A *task* is then created in the server database;

a task is a set of jobs that are created in response to a user submission; in general all the jobs in a task require the same resources and access the same input dataset. The client permits the user to manage all jobs in a task simultaneously. For example, if the user requests that the task be killed or resubmitted, all the jobs associated with that task will be killed or submitted again to the Grid scheduler simultaneously. The client can also monitor the progress of execution and retrieve output and log files.

The server frontend acts as a gateway for user requests and handles user authentication. Users must be registered in SiteDB in order to access the server. The user configuration is validated again and is stored in an Oracle database called the *task database*. All the communications between the client and the server frontend use a REST-ful approach [**?**] which provides a clear separation between the data and the interface used to modify this data, allowing the technology used to store the data to change without having to change the CRAB3 client. Indeed, CRAB3 supports both the Oracle and MySQL backends.

The task worker component is responsible for fetching work from the task database using the REST interface. It implements the submission, kill, and resubmit commands. Many instances of the task worker can run in parallel on different machines, and each one initiates multiple slave processes that process tasks in parallel. Each process is responsible for executing one unit of work on a task. For example, for the submit command the task worker will perform the DBS discovery and job splitting operations mentioned above, and it will send to the HTCondor Scheduler a description of all the jobs in the task, their input, and what they have to execute.

The task database not only stores information about the task and its status, but it is also used to synchronize with the task worker component, making sure multiple instances do not accidentally process the same task.

A script called the job wrapper is sent to each worker node and executed. It 1) sets up the CMS software environment, 2) executes the user code, 3) records monitoring information, 4) reports any errors and 5) transfers the output.

Output file transfer (stageout) is performed in two steps. The output files are first copied to a temporary storage area local to the execution site, and then a component called Asynchronous StageOut (ASO [12]) is notified that the output files need to be copied to the location where the user was provided storage space. ASO then manages the file transfer using a File Transfer Service (FTS [**?**]) server and channels, as in PhEDEx. ASO then updates the file metadata in DBS to reflect the description and location of the new files once they are at the final destination. This is one of the improvements in CRAB3. In CRAB2 the copy of the output files was done directly from the worker node without scheduling by a central server. This resulted at times in congestion, timeouts, and failed transfers, particularly if many jobs tried to write to the same location. Unmanaged transfers could also affect other unrelated data transfer activities. This approach also allows the worker node to return to available status as soon as the job completes, rather than wasting available CPU cycles waiting for the outputs to be staged out.

The DAG description for each job includes a *postjob* script executed on the scheduler machine. The postjob communicates with ASO, waits for the transfer(s) to finish, decides if the job needs to be resubmitted (either because of a job or a transfer failure), and executes cleanup operations. Resubmission is done restarting the DAG node with another submission to the grid pool and a new postjob.

All job submission activities in CMS are monitored by a central service called Dashboard [13].

4. Commissioning and adoption

After an intense period of testing and commissioning by experts and power users, the CRAB3 system was exposed for tests to physicists on the 1$^{\text{st}}$ June 2014. The context was the Computing, Software, and Analysis Challenge (CSA14 [14]), a series of large-scale tests of the complete CMS data processing, software, and analysis chain that was organized from June to September 2014 to assess the readiness of the CMS distributed computing system for the start-up of the LHC Run 2. The whole CRAB3 system, including ASO and the global pool, was part of the test. In particular, functional and scale tests were organized for CRAB3.

The objective of the scale test was to check how the system behaves under heavy load. The targets were set according to the usage observed during Run 1, and consisted of 20k parallel running jobs and 200k completed jobs per day. In order to ensure that the target load was reached, in addition to the user activity, complementary automatic job submission was set up via HammerCloud [15], a stress testing system of distributed job execution.

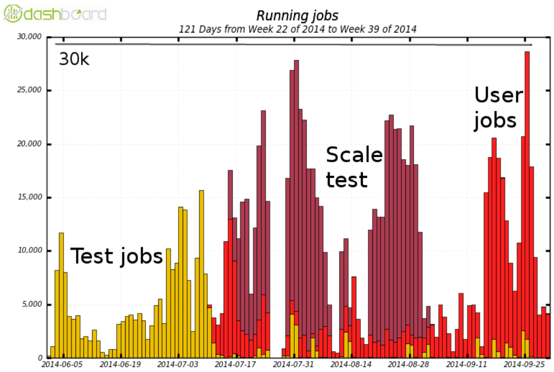

Figure 2. Running jobs during the CSA14 challenge.

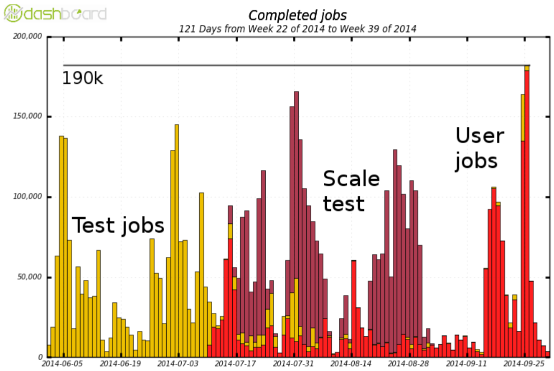

Figure 3. Total daily completed jobs during the CSA14 challenge.

Figures 2 and 3 show that the established targets were achieved. In particular, towards the end of the challenge almost 30k parallel running user jobs were executed in the system and a peak of 190k total daily completed jobs were reached. This demonstrated that the load achieved during LHC Run 1 could be sustained. However, to avoid interfering with user jobs running under CRAB2, the system was not pushed to its limits. Scalability tests of the components show encouraging results; for example the ASO component was able to perform 600k transfers over the three days of the test, twice the expected load [12].

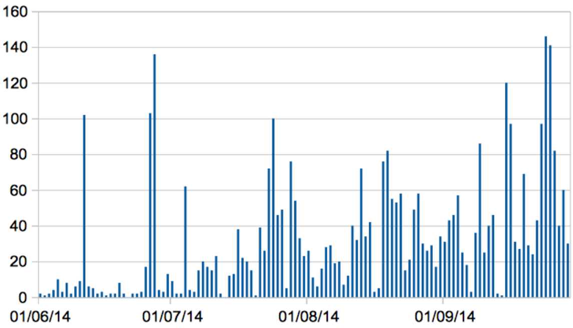

Figure 4. Number of tasks submitted during the CSA14 challenge.

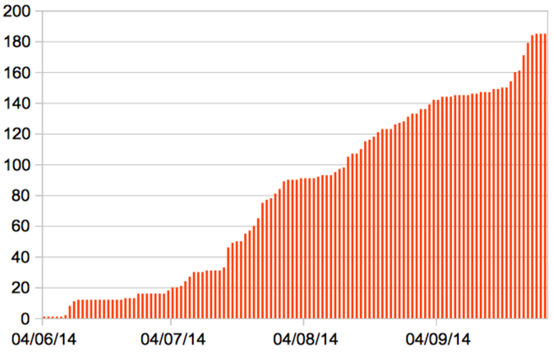

Figure 5. Total miniAOD user produced datasets during the CSA14 challenge.

In addition to the scale tests, functional testing of the activities expected at the beginning of data taking was conducted. The system was exposed to the user community through a miniAOD production campaign, a new reduced CMS data format expected to be used by the majority of data analyses. The objective was to collect feedback from physicists, and start a process of feature-driven development where feedback from users is collected, implemented in CRAB3, and deployed in production in a monthly basis. The code is frozen two weeks before the monthly deadline, and deployed in a testbed where the integration group can perform adequate testing.

During the CSA14 challenge more than 200 users tried the CRAB3 system, submitting up to 35 tasks (a set of related jobs) per day (Figure 4), and more than 180 datasets were published in the CMS bookkeeping system (DBS) in the context of CSA14 user miniAOD production (see Figure 5). Many operational/infrastructural issues were encountered and promptly solved, and the monthly release development cycle, which is still ongoing, proved to be a successful strategy to provide a solid, tested and continuously updated product to the community.

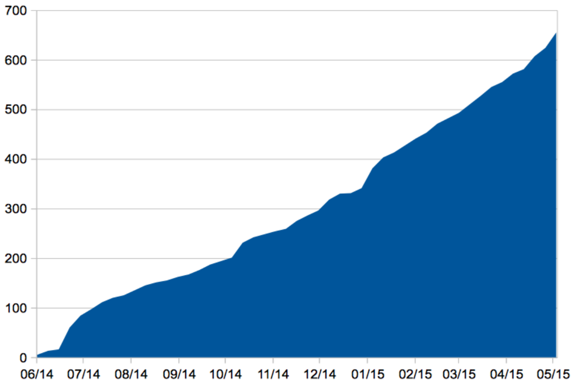

Figure 6. Cumulative CRAB3 users since June 2014.

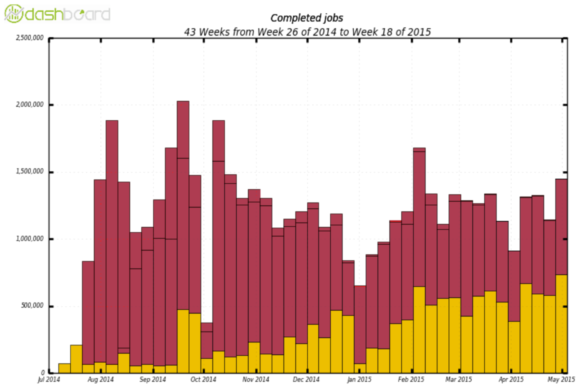

Figure 7. CRAB2 vs CRAB3 jobs since July 2014.

Figure 6 shows that the cumulative number of users since the system was declared production ready is increasing. Each point corresponds to the number of users who submitted at least one task up to a certain date. At the time of writing 665 distinct users have submitted at least one task to the CRAB3 system. Figure 7 shows that the number of CRAB3 jobs is increasing while the number of CRAB2 jobs is decreasing.

The final release of CRAB2 was issued in November 2014, and support is now limited to the few areas which are not yet supported under CRAB3. CRAB2 applications continue to run, but users are encouraged to make the transition as soon as feasible and the majority have already done so.

5. Operations and evolution

CRAB3 introduces many improvements also in the area of the daily operations activities. Over time, the CRAB2 code had become complex and difficult to maintain. For example, the scheduling function under CRAB2 required plug-in software to be developed on an ad-hoc basis to provide access to local resources that were not accessible under the normal Grid access procedures. Under CRAB3, the HTCondor scheduler takes over many of these functions, keeping the CRAB code simpler and more maintainable.

Additionally, the usage of a shared glideinWMS pool (the Global Pool [7]) for both user data analysis and central production of datasets and the usage of a common infrastructure for transfers based on FTS [?] significantly reduces the effort needed to mantain the submission and transfer infrastructures. This not only improves the long term sustainability of CRAB3, but also of the whole CMS computing software stack in general.

The development effort has now shifted to adding new functionalities which were not available under CRAB2. For example, an option has been added to estimate time required by the user code per event analyzed; this can be helpful in setting splitting parameters. In addition, a library of functions provides access to the client from within a python script, allowing job submission and other CRAB operations to be automated within a complex user workflow.

Future plans for CRAB3 include more flexible management of user output data, especially related to the possibility to move them around with PhEDEx and Tier2 disk space allocated to groups; low-latency file handling solutions when user output is small; long term persistency and traceability of the CRAB3 configuration, possibly in the context of analysis documentation and preservation; and of course constant integration with the evolving CMS software infrastructure.

6. Conclusions

CRAB3 brings to CMS analysis users a client-server architecture which allows them to transparently deal with the intrinsic unreliability of the large distributed computing infrastructure. It addresses the main weaknesses of the past tool in the output management area and largely improves on friendliness and usability.

In this new implementation we focused on clear division into components, limiting as much as possible the in-house development in favor of adopting standard solutions like HTCondor and FTS. We also managed to use the same implementation of the underlying Grid submission infrastructure (a global HTCondor pool implemented via glideinWMS), thus largely simplifying the operational load.

Offering a long, smooth transition and a predictable release schedule is proving very effective for a successful migration of our large user community. The vast majority of users report that they are much happier with this new version of CRAB and do not return to the previous version. Overall CRAB3 deployment has been a success. Because of extensive testing in 2014 with real use cases, the support is a lighter activity than it was for CRAB2 at a similar stage of maturity.

Acknowledgments

The present work is partially funded under program PRIN 20108T4XTM.

References

[1] Worldwide LHC Computing Grid `http://wlcg.web.cern.ch/`

[2] Giffels M, Guo Y and Riley D 2014 *Journal of Physics: Conference Series* **513** 042022 URL `http://stacks.iop.org/1742-6596/513/i=4/a=042022`

[3] Egeland R, Wildish T and Metson S 2008 *PoS* **ACAT08** 033

[4] Metson S, Bonacorsi D, Ferreira M D and Egeland R 2010 *Journal of Physics: Conference Series* **219** 072044 URL `http://stacks.iop.org/1742-6596/219/i=7/a=072044`

[5] Fajardo E, Gutsche O, Foulkes S, Linacre J, Spinoso V, Lahiff A, Gomez-Ceballos G, Klute M and Mohapatra A 2012 *Journal of Physics: Conference Series* **396** 042018 URL `http://stacks.iop.org/1742-6596/396/i=4/a=042018`

[6] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Wurthwein F 2009 The Pilot Way to Grid Resources Using glideinWMS *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 02* CSIE '09 (Washington, DC, USA: IEEE Computer Society) pp 428–432 ISBN 978-0-7695-3507-4 URL `http://dx.doi.org/10.1109/CSIE.2009.950`

[7] Gutsche O, Letts J, Balcas J, Belforte S, Bockelman B, Wissing C, Larson K, Mascheroni M, Mason D, Mc Crea A, Saiz Santos M, Sfiligoi I, Hufnagel D, Piperov S, Colling D and Khan F 2015 Using the glideinwms system as a common resource provisioning layer in cms to be published in the proceeds for CHEP2015: 21st International Conference on Computing in High Energy and Nuclear Physics

[8] Kenneth Bloom and the CMS Collaboration 2014 *Journal of Physics: Conference Series* **513** 042005 URL `http://stacks.iop.org/1742-6596/513/i=4/a=042005`

[9] Bauerdick L A T, Bloom K, Bockelman B, Bradley D C, Dasu S, Dost J M, Sfiligoi I, Tadel A, Tadel M, Wuerthwein F, Yagil A and the Cms collaboration 2014 *Journal of Physics: Conference Series* **513** 042044 URL `http://stacks.iop.org/1742-6596/513/i=4/a=042044`

[10] CMSSW Application Framework `https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCMSSWFramework`

[11] Blomer J, Aguado-Sánchez C, Buncic P and Harutyunyan A 2011 *Journal of Physics: Conference Series* **331** 042003 ISSN 1742-6596 URL `http://stacks.iop.org/1742-6596/331/i=4/a=042003?key=crossref.f4f69e2c25fda333e912ab8515e3179e`

[12] Riahi H e a 2015 Asyncstageout: Distributed user data management for cms analysis to be published in the proceeds for CHEP2015: 21st International Conference on Computing in High Energy and Nuclear Physics

[13] Karavakis E, Andreeva J, Khan A, Maier G and Gaidioz B 2010 *Journal of Physics: Conference Series* **219** 072038 URL `http://stacks.iop.org/1742-6596/219/i=7/a=072038`

[14] Computing, Software, and Analysis Challenge 2014 `https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCSA14`

[15] Elmsheuser J, Llamas R M, Legger F, Sciabà A, Sciacca G, beda Garca M and van der Ster D 2012 *Journal of Physics: Conference Series* **396** 032111 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032111`