Journal of Physics: Conference Series **513** (2014) 042022 doi:10.1088/1742-6596/513/4/042022

Data Bookkeeping Service 3 - Providing event metadata in CMS

M Giffels[1], Y Guo[2] and D Riley[3]

[1] PH-CMG-CO. CERN, CH-1211 Genève 23, Switzerland
[2] Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.
[3] Cornell University, Ithaca, New York, U.S.A.

E-mail: Manuel.Giffels@cern.ch, yuyi@fnal.gov, Daniel.Riley@cornell.edu

Abstract. The Data Bookkeeping Service 3 provides a catalog of event metadata for Monte Carlo and recorded data of the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) at CERN, Geneva. It comprises all necessary information for tracking datasets, their processing history and associations between runs, files and datasets, on a large scale of about 200,000 datasets and more than 40 million files, which adds up in around 700 GB of metadata. The DBS is an essential part of the CMS Data Management and Workload Management (DMWM) systems [1], all kind of data-processing like Monte Carlo production, processing of recorded event data as well as physics analysis done by the users are heavily relying on the information stored in DBS.

1. Introduction

The CMS Data Bookkeeping Service (DBS) provides databases and services to store and access metadata related to the CMS physics event data. Besides the record of what data exists, DBS also contains process-oriented provenance information, including parentage relationships of files and datasets, configurations of processing steps, as well as associations with run numbers and luminosity sections to find any particular set of events within a dataset.

The current version of DBS, DBS 2 [2] was designed and implemented in 2006-2007, before the operation of the LHC. At that time, CMS did not have a standardized service architecture for the implementation and operation of its web services, so the DBS 2 was implemented using Java servlets in an Apache Tomcat container and XML RPC for the client-server communication. In some cases DBS 2 has very "thick" client APIs, which eventually lead to numerous problems with API versioning and scalability issues.

Since the time DBS 2 has been designed, the CMS data processing model has evolved in ways not anticipated by the DBS 2 design, so many requests were made for additional data to be stored in DBS 2 that were not entirely consistent with its original purpose. Furthermore, the CMS DMWM project had developed a standard architecture and deployment model for CMS web services. A project review in 2009 led to the decision to re-design the DBS, to better match the CMS data processing model and to better integrate with other DMWM projects. The design goals for DBS 3 included: alignment of the data model with the evolved CMS processing model; spinning-off any services that were outside the project scope; simplified and lightweight APIs and optimized database schemas based on observed DBS 2 usage, "logical" provenance,

collapsing out the split and merge steps of the processing history that have no effect on the data content. An initial set of high and low level APIs, and an initial architecture and DB schema were developed over Summer 2010. After an intensive design review in September 2010, prototypes of various possible architectural and technology choices were built, resolving in the current DBS 3 design and implementation.

2. The design of RESTful DBS 3 Web service

The CMS DMWM project has standardized on the Representational State Transfer (REST) architecture [3] for its web services. This simplifies the integration of the different components of the DMWM system into a common architecture. REST also imposes the discipline of thin clients. The DMWM project largely standardized on the Java Script Object Notation (JSON) data-format for client-server communication, as a lightweight alternative to XML RPC used in the previous generation of DBS. The common implementation language for DMWM web services has become Python, using SQLAlchemy and CherryPy toolkits, as well as a common "WMCore" REST framework. During prototyping it turned out, that the standardized technologies are competitive with the performance and reliability of the Java servlets used in DBS 2, while the advantages of using a common language and toolkit across all DMWM projects are obvious. By adopting the standardized DMWM service architecture, DBS 3 transparently becomes a data service provider for the CMS Data Aggregation Service (DAS) [4], which makes it possible to de-aggregate DBS into several services with narrower scope without significant loss of query functionality.

DBS 3 has been completely re-designed and re-implemented in Python utilizing the CMS DMWM standard for web services described above. DBS 3 has been implemented as RESTful interface (see Figure 1). The API is chosen by the path in the URI and the corresponding operation by the HTTP method used. DBS 3 supports GET, POST and PUT operations. The deletion of data inside the catalog is not supported to ensure perpetual traceability. The client-server communication is stateless, which enhances the scalability of the service. Being deployed on the CMSWEB cluster [5], also part of the CMS DMWM project, DBS 3 is taking advantage of standardized deployment and operation procedures, and authentication using X509 certificates provided by the CMSWEB Apache front-ends. To persistently store the metadata information, an Oracle Database backend provided for CMS [6] is utilized.

The database schema for DBS 3 has been completely revised, based on the use cases and usage statistics for DBS 2, and the feedback gathered during the design review. One important change in the DBS 3 schema was denormalization. The DBS 2 schema was fully normalized, leading to excessive table joins and lock contention. In DBS 3 some tables were removed, as well as some relationships that were better modeled outside of the DBS schema. These changes sped up searches and also improved the insertion of data by removing foreign keys and lock contention. In addition, the DBS 3 schema particularly benefitted from the narrower and more precisely defined project scope compared to DBS 2. The narrower scope was largely made feasible by the integration of different services through the common DMWM architecture.

3. Mastering the transition to DBS 3

The CMS DBS is a federated system with several different DBS instances used for specific purposes. The underlying principle in CMS is the separation of official and user-created data. The Global-DBS contains metadata with respect to all official CMS data, real or simulated, that are available for physics analysis, whereas the metadata of user-created datasets is stored in the physics analysis DBS instances. Two of them are available for individual physicists or physics groups to record and manage the results of their analysis steps. Besides those instances there are also a CAF instance, that records data from the prompt reconstruction stream used for detector

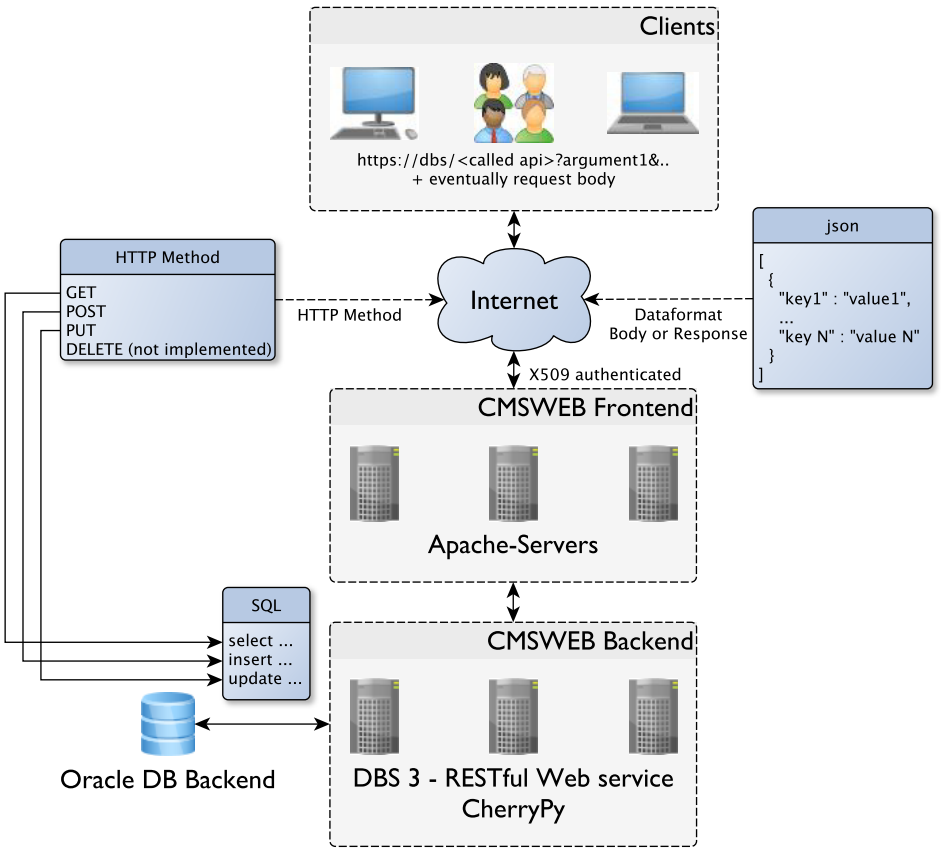

Figure 1. Function principle of the DBS 3 RESTful web service deployed on the CMSWEB cluster.

calibrations and diagnostics, and a CMS Tier0 DBS instance, that records information for data from the detector as it is processed by the Tier0 facility.

As shown in figure 2 shown, many services depend on the information stored in DBS.

- The DAS is using DBS as one of its data providers. DAS is fetching the necessary information based on user inputs from DBS and displaying them, aggregated with information from other data providers neatly arranged in a Web GUI.
- The WMAgent, the tool used for distributed data processing of official data in CMS heavily depends on DBS.
- The StoreResults service [7] is repacking and migrating physics analysis data determined to be of general use for a physics group and fulfilling the same requirements as official data to Global-DBS and the CMS data transfer system PhEDEx [8].
- The CMS Remote Analysis Builder 2 (CRAB 2) [9], which is the CMS tool for distributed user analysis.

DAS and WMAgent are already supporting both systems, DBS 2 and DBS 3. WMAgent is also supporting simultaneously writing to both system. The StoreResults service and CRAB 2 are not able to talk to DBS 3 and it is currently not planned to add support for DBS 3 to those services. The reason for that is the ongoing development of new version CRAB 3 and

StoreResults@WMAgent having native support of DBS 3. The transition to DBS 3 turns out to be a complex endeavor.

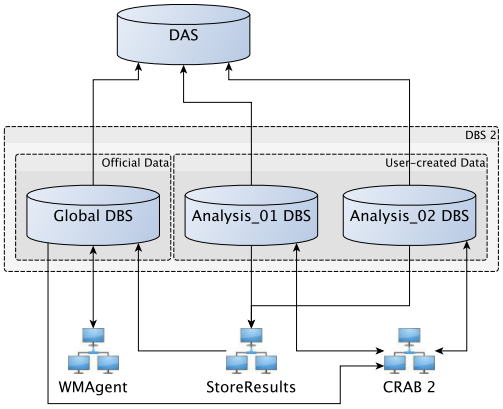

Figure 2. The dependencies of different CMS Data Management and Workload Management services on the DBS 2.

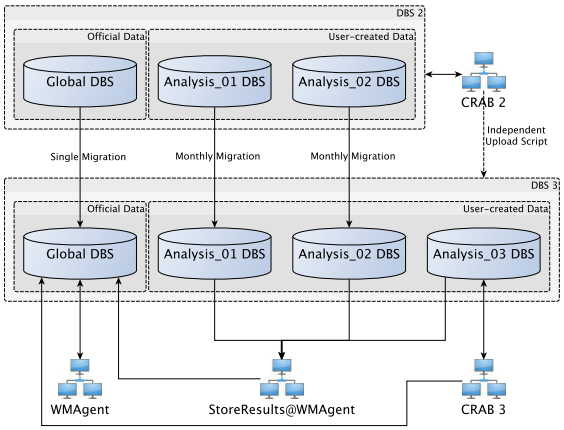

Figure 3. The DBS 2 to DBS 3 transition plan and the dependent services.

To tackle the transition a detailed plan depicted in figure 3 has been developed. The Global-DBS will be migrated only once at the desired date of transition and Global-DBS in DBS 2 will become read-only for legacy reasons. The two physics analysis DBS instances in DBS 2 will be migrated periodically once a month and the corresponding DBS 3 counterpart will be operated in read-only mode. In addition, a new physics analysis DBS instance will be created in DBS 3. The reason for that is the planned smooth transition between CRAB 2 and CRAB 3. Both system will be operated in parallel for a certain time. The WMAgent, CRAB 3 and StoreResults@WMAgent will use DBS 3 only, whereas CRAB 2 will continue to use DBS 2. Additionally, an independent upload script will allow CRAB 2 users to publish their datasets in the newly created physics analysis DBS in DBS 3. For the remaining two DBS instances not depicted in the figure 3 the migration plan is the following. The CMS Tier0 DBS instance is not needed anymore due to the changed data processing model in CMS, whereas the CAF instance will be migrated only once at the target date similar to Global-DBS.

The migration to DBS 3 is planned for end of 2013, however all dependent services need to be ready for DBS 3 before the migration can happen, so that an exact target date can not yet be determined. The migration itself is a challenging task, DBS 2 is growing very fast and currently comprises 40 million files, 240,000 blocks and 200,000 datasets and the database schema of DBS 3 is completely different, so that a data conversion is necessary. The adaption of the metadata stored in DBS 2 into the new database schema of DBS 3 is done by PL/SQL scripts. Once the data has been transformed, a one-to-one validation is performed to ensure data consistency. This task is driven by a Python script using standard SQL statements. The migrations always happens "en block", an incremental migration of data is not provided.

To ensure a smooth transition between both systems, two migration pilot runs have been performed. The Global-DBS of DBS 2 has been operated in read-only mode during the migration. The first migration pilot run took place in March 2013. After the one-to-one validation of the content, the WMAgent started to insert new metadata for the Heavy Ion runs of CMS into both systems. Another migration pilot run has been done in May 2013. Since that time all WMAgents are publishing all metadata of officially processed data in both systems, in average about 18,000 blocks per month. To discover potential inconsistencies between DBS 2 and DBS 3, monthly consistency checks are performed. So far, only minor and well-understood

differences occurred. Both migration pilot runs revealed that the migration and validation of data strongly depend on the load on the Oracle database. However, the goal to perform the migration and validation in less than a day was archieved in both cases.

4. Performance testing of DBS 3 using the PhEDEx LifeCycleAgent

Besides the numerous tests performed during development, deployment and validation, the performance testing of DBS 3 is a further step to ensure a reliable system even in conditions of high load. The LifeCycleAgent is perfectly suited for the DBS 3 performance tests. It was originally developed by the PhEDEx team to drive a realistic simulation of their data transfer system [10]. The LifeCycleAgent can execute functions in perl modules as well as external scripts, which also makes it a handy tool for other projects. The way of working of the LifeCycleAgent is shown in figure 4. The LifeCycleAgent uses a configurable number of worker threads to process individual tasks fetched from a task queue. Each task is described by a so called payload in JSON format. The payloads can be modified within the tasks depending on its result. The modified payloads are then passed from one task to another, which enables the possibility to dynamically add tasks to the queue. The interrelationship between the tasks is defined in a so called workflow.

For the DBS 3 performance tests, three such workflows have been defined to model a realistic simulation of eventual use-cases. The DBS 3 access pattern of CRAB 3 performing distributed user analysis (figure 5), the DAS WebGUI access pattern (figure 6) and insertion of blocks by WMAgent performing event generation or event processing tasks (figure 7). All those workflows start with an initial Python script creating the entire payloads for the performance test by forking single tasks. For the analysis all Analysis Object Data (AOD) datasets in DBS are listed and a second Python script simulates the access of CRAB 3 to DBS 3. This workflow is a realistic simulation of N analysis users creating and submitting their analysis jobs to the Grid. In the DAS workflow real user queries are obtained by parsing DAS logfiles and a task per query is forked from the initial script. To simulate blocks insertion by WMAgent a fake data provider is creating blocks to be inserted into DBS according to the configuration. All workflows were executed on a dedicated queue on the CERN batch system (4 hosts, 8 job slots each) and ran against the cmsweb pre-production cluster.

One result of the performance test that has a major impact is the difference between using urllib2 and PycURL in a DBS 3 client. As shown in figure 8 the average client response time in the CRAB 3 test case can be decreased by roughly a factor of 20 using PycURL (figure 9). The reason for that is the following, CRAB 3 is calling many DBS 3 APIs in a row. In contrast to urllib2, PycURL is using ssl authentication caching, that means the same connection can be re-used without re-doing the ssl authentication handshake with the frontend. The same effect is visible also in the CPU utilization of the Apache frontend. Figure 10 shows high CPU utilization of the frontend using a urllib2 based client, whereas the CPU is drastically decreased when using a PycURL based client (figure 11).

To test the writing performance of DBS 3, we simulated a heavy, but realistic production task. For the LHE (Les Houches Event Files) [11] event generation computing operations inserts blocks containing many files, which again contain a large number of luminosity sections. That was usually a very expensive operation for DBS 2 and the insertion of one block took in average about 4 hours. The same insertion in DBS 3 takes only about 50 seconds per block, which is also a huge improvement.

In addition to testing each use-case standalone, combined reading and writing tests have been performed, too. The tests simulated 40 servers reading in parallel and 16 servers writing every 60 seconds a new block with an average number of files and luminosity sections. That corresponds to the current injection rate into DBS 2. CMS currently uses about 16 servers, that will read and write to DBS 3, so even under high load DBS performs well.

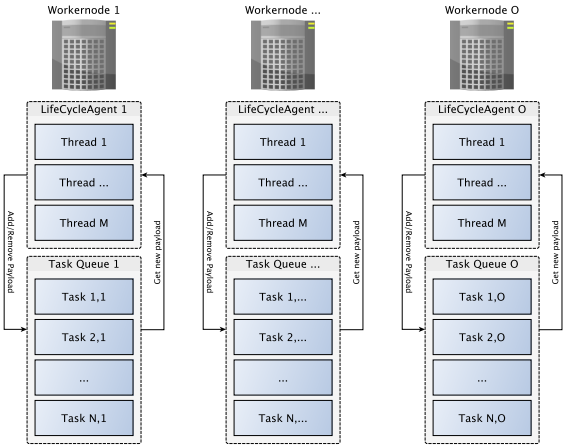

Figure 4. Sketch of the LifeCycleAgent working principle.

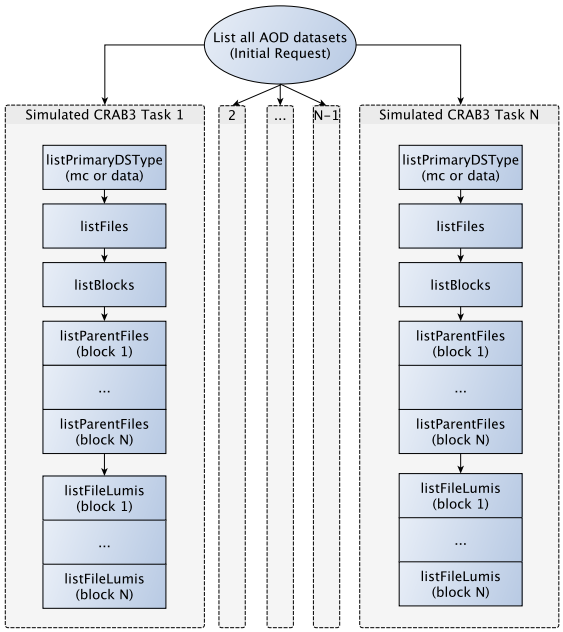

Figure 5. Realistic example of an analysis workflow defined in a LifeCycleAgent workflow.

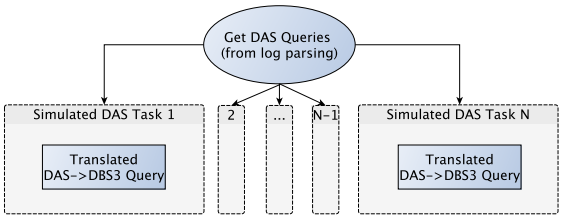

Figure 6. Realistic example of an DAS workflow defined in a LifeCycleAgent workflow.

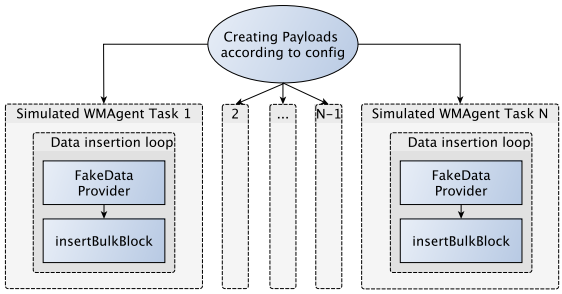

Figure 7. Realistic example of an processing workflow defined in a LifeCycleAgent workflow.

5. Conclusion

The design and development of DBS 3 was greatly influenced by the lessons learned from its predecessor and the ongoing revision of the DMWM software. DBS 3 now better fits the needs of the current data processing model of CMS and is better integrated into the realm of DMWM. The RESTful design using lightweight APIs and a stateless client-server communication ensures a better scalability of DBS 3. The transition between DBS 2 and DBS 3 turned out to be a complex endeavor, however a reliable method of the doing the migration has been developed and two migration pilot runs finished successful and helped to find problems before going officially into production. Realistic simulations of the day to day access pattern to DBS 3 have shown that the performance of DBS 3 is in a good shape and in particular the writing performance has been improved significantly. Before going in production end of 2013, it is planned to do another migration pilot run and to configure DAS to query DBS 2 and DBS 3 in parallel, which will move DBS 3 a bit closer to production.

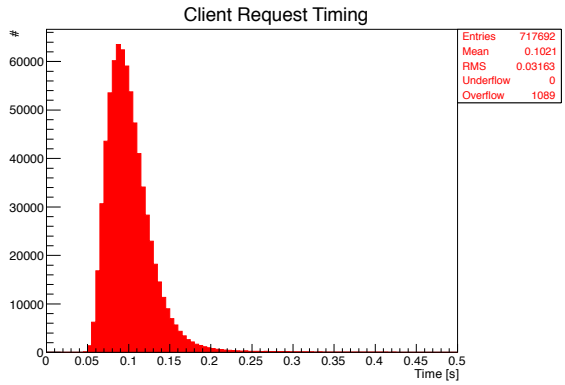

Figure 8. The client request timing using Python's urllib2 to connect to the server.10 threads are accessing cmsweb-testbed in parallel.

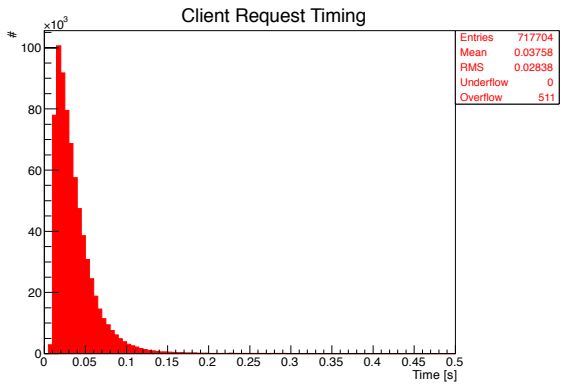

Figure 9. The client request timing using PycURL to connect to the server.10 threads are accessing cmsweb-testbed in parallel.

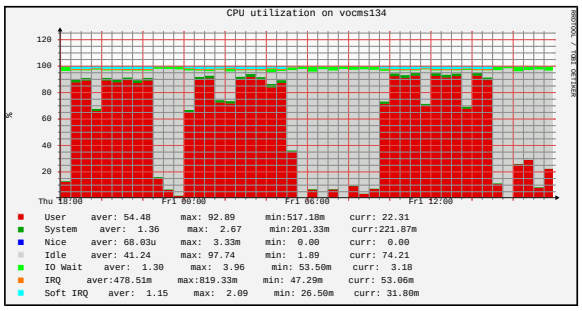

Figure 10. The CPU utilization of the Apache frontend, utilizing Python's urllib2 to connect to the server (10 threads in parallel).

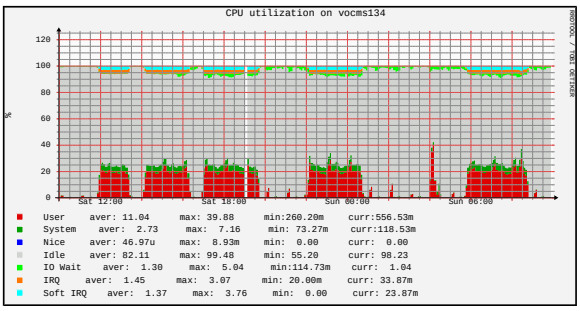

Figure 11. The CPU utilization of the Apache frontend, utilizing PycURL to connect to the server (10 threads in parallel).

References

[1] Widlish T et al, 2013, The CMS Data Management System, in this proceedings
[2] Afaq A et al, 2008, The CMS Dataset Bookkeeping Service, *J. Phys. Conf. Ser.* **119** 072001
[3] Fielding R T, 2000, Architectural Styles and the Design of Network-based Software Architectures, Dissertation, *University of California, Irvine*, ISBN: 0-599-87118-0
[4] Ball G et al, 2011, Data Aggregation System: A system for information retrieval on demand over relational and non-relational distributed data sources, *J. Phys. Conf. Ser.* **331** 042029
[5] Metson S et al, 2008, CMS offline web tools, *J. Phys. Conf. Ser.* **219** 082007
[6] Pfeiffer A et al., 2012, CMS experience with online and offline databases, *J.Phys.Conf.Ser.* **396**
[7] Giffels M, 2011, et al, Design and early experience with promoting user-created data in CMS, *J. Phys. Conf. Ser.* **331** 072049
[8] Egeland R et al, 2010, PhEDEx data service, *J. Phys. Conf. Ser.* **219** 062010
[9] Spiga D et al, 2007, The CMS Remote Analysis Builder (CRAB), *Lect. Notes Comput. Sci.* **4873** 580-586
[10] Wildish T et al, From toolkit to framework - the past and future evolution of PhEDEx, *J.Phys.Conf.Ser.* **396**
[11] Alwall J et al., 2007, A Standard format for Les Houches event files, *Comput.Phys.Commun.* **176** 300-304