Experience in CMS with the Common Analysis Framework Project

Marco Mascheroni

INFN Milano Bicocca

On behalf of the CRAB3-PanDA Team

CHEP, Amsterdam, October 2013
Outline

1 Motivation

2 History
   - The feasibility study
   - Proof of Concept

3 Technical Overview
   - Global Software Architecture
   - CMS Components
   - Common Components
   - Deployment

4 Conclusions
Motivation

- After 2 years of successful LHC data taking, processing, and analysis, we are now at LS I: opportunity to try to optimize LHC computing.
- **Sustainability** is going to be a crucial aspect as experiments are steadily decreasing their development effort.
- In the long run, **Common solutions** can be used to optimize the development effort, and reduce the maintenance and support costs of a tool.
- Several ”multi-experiment” tools have been proposed and are widely used by the LHC community: popularity, site cleaning agent, dashboard, hammer cloud, etc\(^1\)

The feasibility study

- Even though LHC experiments have very similar workflows, common solutions among high level submission tools have never been developed.
- Each experiment has its own workflow manager.

The idea

A feasibility study on possible approaches to create a common analysis framework based on PanDA

- Started in March 2012
- Involving the experiment support group at CERN, ATLAS and CMS
- Results presented at CHEP 2012
As expected there were many similarities in the analysis use case of the two experiments:

- Client tool which, given the user’s requirements, generate the jobs, submit them to a job manager.
- Job manager which is able to track the job, submit and kill them, etc.

Core workflow engine suitable candidate for common solution!
Proof of concept

The objective

To develop a common system for submitting analysis jobs to the distributed infrastructure

Based on the PanDA software

- ATLAS Production and Distributed Analysis job manager
- Able to handle 1M jobs per day (enough for CMS which only require 200k jobs)
- Stable product used from many years by ATLAS

Outcome

December 2012: first basic CMS proof of concept workflow run on the system!
Consolidation of the proof of concept prototype

Based on the following components:

For data management:

- **AsyncStageout**
  - **APF**
  - **PandaServer**
  - **PandaMon**

For data management:

- **CRAB3 Client**
- **REST**
- **Task Worker**

**Experiment Specific**

**Shared Components**

**Grid/Cloud Infrastructure**

Mostly integration of already existin component
Components overview

- **Components Overview**
  - **CRAB3 Client**
    - PUT Workflow
    - POST workflow (resubmit)
    - DELETE workflow (kill)
    - GET Workflow status (out/log) (report) (etc)
  - **Oracle Task Database**
    - Workflow
    - Metadata
  - **Task Worker**
    - upload user executables
    - lock work
    - get work
    - set result
    - PUT filemetadata (Coming from CMS Adder Component)

Marco Mascheroni (INFN Milano Bicocca)
Version 3 of the CRAB client

- Command Line Interface for the user (python tool)
  - Commands: submit, status, getout, getlog, report
- Lightweight stateless interface
  - Basically just pycurl and python2.6 required
- Modular and pluggable
  - New command or new Jobtype? Just subclass the proper interface and code!
REST

CRAB3 REST Api

- Act as a gateway for user requests
  - Handles authentication through https (X509 certificates)
- Validate user requests, cache them in the Oracle DB
- Easy to script against
- REpresentational State Transfer interface
  - Resources in the DB accessible through 4 HTTP verbs
  - Clear separation data and interface (easy to switch DBMS)
- Designed to support multiple JobManager
  - e.g.: glideInWMS in addition to PanDA
TaskWorker

- Component that stands between REST and PanDA
- Can be easily adapted to other JobManager
- Multi-threaded and distributed architecture

- Work types: Submit, Kill, Resubmit
- Process to execute set of Actions associated to the work

E.g.:

<table>
<thead>
<tr>
<th>Work: Submission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act1: Data Loc. Discovery</td>
</tr>
<tr>
<td>Act2: Splitting</td>
</tr>
<tr>
<td>Act3: Brokerage</td>
</tr>
<tr>
<td>Act4: Injection</td>
</tr>
</tbody>
</table>
Components overview

1) Ask for jobs
2) Update with job progresses
PanDA Server

Functionalities
- Takes JobSpecs and store them in the oracle DB
- Handles queues, job scheduling and job priority

Comments
- Using the same code as ATLAS. No branching!
- Oracle DB cloned from the PanDA production database
- Same job state machines for the jobs
- Plugged Data Management parts (CMSAdderComponent)
AutoPilot Factory

Functionalities

- APF sends and manage pilots based on the information in the Panda Server (uses condor.g)

Comments

- Light and easy to operate.
- From the CMS side just a small patch to apply (to handle pool of credentials)
Pilot

Functionalities

- Contact PandaServer and get jobs to execute
- Report to the Panda Server info about the job

Comments

- Required some development
- Original pilot modified to allow execution of specific experiment code
- Transformation and stageout are experiment specific plugins
The CMS testbed

In August CMS finished the deployment of its own instance of the framework:

- 10 machines installed for all services
- 47 CMS site added to the APF configuration
- Configuration of the APF done through the ATLAS Grid Information System
Open issues

The testbed showed it was possible to use a common solution between the experiments. However:

- PanDA server still in transition from experiment specific to a service
  - Code installed directly from SVN
  - There are no DB create.sql scripts. Only cloning is possible.
  - Database uses ATLAS naming convention
- No site-level user traceability
  - User code is executed with the pilot credentials, no glExec (although there’s work in progress)
- Not a clear separation between scheduling algorithm, and source code
  - Policies cannot be given as an external configuration
Conclusion and future work

Conclusions

- Feasability study highlighted similarities between ATLAS and CMS analysis workflow engines
- Demonstrated it was possible use a common framework with a proof of concept
- Consolidated of the proof of concept and deployed a working testbed

Future work

- Demonstrate the viability of the design concept of multiple workflow
  - Integration of the tesbed components with the CMS production workflow system
- Perform a scale test
- HammerCloud integration
Acknowledgments

This work was possible thanks to the contributions of:

- **CERN-IT:**
  - Mattia Cinquilli, Domenico Giordano, Alessandro Di Girolamo, Maria Girone, Niccolò Magini, Valentina Mancinelli, Daniele Spiga

- **CMS:**
  - Tommaso Boccali, Daniele Bonacorsi, Federica Fanzago, Ian Fisk, Jose Hernandez, Preslav Konstantinov, Hassen Riahi, Lola Saiz Santos, Eric W Vaandering

- **ATLAS:**
  - Velerie Emil Fine, Tadashi Maeno, Paul Nilsson