Contribution ID: **63** Type: **Oral contribution**

BOSS: the CMS interface for job summission, monitoring and bookkeeping

Thursday 2 March 2006 15:00 (30 minutes)

BOSS (Batch Object Submission System) has been developed in the context of the CMS experiment to provide logging and bookkeeping and real-time monitoring of jobs submitted to a local farm or a grid system. The information is persistently stored in a relational database (right now MySQL or SQLite) for further processing. In this way the information that was available in the log file in a free form is structured in a fixed-form that allows easy and efficient access. The database is local to the user environment and is not requested to provide server capabilities to the external world: the only component that interacts with it is the BOSS client process.
BOSS can log not only the typical information provided by the batch systems (e.g. executable name, time of submission and execution, return status, etc···), but also information specific to the job that is being executed (e.g. dataset that is being produced or analyzed, number of events done so far, number of events to be done, etc···). This is done by means of user-supplied filters: BOSS extracts the specific user-program information to be logged from the standard streams of the job itself filling up a fixed form journal file to be retrieved and processed at the end of job running via the BOSS client process.
BOSS interfaces to a local or grid scheduler (e.g. LSF, PBS, Condor, LCG, etc···) through a set of scripts provided by the system administrator, using a predefined syntax. This allow hiding to the upper layers its implementation details, in particular whether the batch system is local or distributed. The interface provides the capability to register, un-register and list the schedulers. BOSS provides an interface to the local scheduler for the operations of job submission, deletion, querying and output retrieval. At output retrieval time the information in the database is updated using information sent back with the job.
BOSS provides also an optional run-time monitoring system that, working in parallel to the logging system, collects information while the computational program is still running, and presents it to the upper layers through the same interface. The real-time information sent by the running jobs are collected in a separate database server, the same real-time database server may support more than one BOSS database. The information in the real-time database server has a limited lifetime: in general it is deleted after that the user has accessed it, and in any case after successful retrieval of the journal file. It is not possible to use the information in the real-time database server to update the logging information in the BOSS database once the journal file for the related job has been processed.
The run-time monitoring is made through a pair client-updater registered as a plug-in module: they are the only components that interact with the real time database. The real-time updater is a client of the real-time database server: it sends the information of the journal file to the server at pre-defined intervals of time. The real-time client is a tool used by BOSS to update his database using the real-time information.

The interface with the user is made through:
a command line , kept as similar as possible to the one of the previous versions; it is the minimal way to access BOSS functionalities to give a straightforward test and training instrument;
C++ API, increasing functionalities and ease-to-use for programs using BOSS: currently it is under development and is meant to grown-up with the users requirements;
Python API, giving almost the same functionalities of the C++ one, plus the possibility to run BOSS from a python command line.
User programs may be chained together to be executed by a single batch unit (job). The relational structure supports not only multiple programs per job (program chains) but also multiple jobs per chain (in the event of job resubmission). Homogeneous jobs, or better "chains of programs", may be grouped together in tasks (e.g. as a consequence of the splitting of a single processing chain into many processing chains that may run in parallel). The description of a task is passed to BOSS through an XML file, since it can model its hierarchical structure in a natural way.
The process submitted to the batch scheduler is the BOSS job wrapper. All interactions of the batch scheduler to the user process pass through the BOSS wrapper. The BOSS job wrapper starts the chosen chaining tool, and optionally the real-time updater. An internal tool for chaining programs linearly is implemented in BOSS but in future external chaining tools may be registered to BOSS so that more complex chaining rules may be requested by the users. BOSS will not need to know how they work and will just pass any configuration information transparently down to them.
The chaining tool starts a BOSS "program wrapper"for each user program.The program wrapper starts all processes needed to get the run-time information from the user programs into the journal file. This program wrapper is unique and it has to be started passing only one parameter, the program id.
The BOSS client determines finished jobs by a query to the scheduler. It retrieves the output for those jobs and uses the information in the journal file to update the BOSS database.
The BOSS client pops the information about running jobs from the real-time database server through the client part of the registered Real Time Monitor. It also deletes from the server the information concerning jobs for which the BOSS database has already been updated using the journal file. The information extracted from the real-time database server may be used to update the local BOSS database or just to show the latest status to the user.

Authors: MACEVOY, Barry (Imperial College London); GRANDI, Claudio (Universita di Bologna); COLLING, David (Imperial College London); CODISPOTI, Giuseppe (Universita di Bologna); CAPILUPPI, Paolo (Universita di Bologna); WAKEFIELD, Stuart (Imperial College London); BACCHI, William (Universita di Bologna); ZHANG, Yong-Jun (Imperial College London)

Presenter: CODISPOTI, Giuseppe (Universita di Bologna)

Session Classification: 2a: Workload management and Workflows

Track Classification: Workload management and Workflows