THE CAPONE WORKFLOW MANAGER

M. Mambelli*, R.W. Gardner**, The University of Chicago, USA
J. Gieraltowski#, Argonne National Laboratory, USA

Abstract

We describe the Capone workflow manager that was designed to work for Grid3 and the Open Science Grid (OSG) infrastructure. It has been used extensively to run ATLAS managed and user production jobs during the past year while undergoing major redesigns to improve reliability and scalability as a result of lessons learned.

This paper first introduces the new design features that cover job management, monitoring, troubleshooting, debugging and job logging. Next, the modular architecture which implements several key evolutionary changes to the system is described: a multi-threaded pool structure, checkpointing mechanisms, and robust interactions with external components; each of these were developed to address scalability and state persistence issues uncovered during operational running of the production system. Finally, we provide results from benchmark stress tests, and compare Capone with other workflow managers in use for distributed production systems.

INTRODUCTION

ATLAS (A Toroidal LHC ApparatuS) [1] is being built by a physics collaboration involving many institutions worldwide. Part of its computing effort, "offline computing", consists in simulating the behavior of the particle detector that will use the Large Hadron Collider at CERN (Geneva, Switzerland) beginning in 2007, and in reconstructing and analyzing the output data. The activity of offline production is organized in data challenges [3,9] where the infrastructure is tested and stressed with increasing volumes of processing.

The Capone project was started in 2004 to deliver a workflow manager that could handle the managed production of ATLAS jobs first on Grid3 [5] and then on its evolution, OSG [13] which are grids composed of computing and storage resources from several universities and national laboratories mainly in the USA. Capone was developed primarily in Python to allow for rapid prototyping. Capone managed the processing of more than 250,000 events in DC2 and Rome production exercises [3] and has been used for user production and some analysis tests.

Another goal the project has been to demonstrate development and support procedures. Capone was already available as a package installable with a single command line command using OSG's installer. With the new version the project branched into separate releases: a stable production release (providing only bug fixes) and a development release that included changes and new features that became the next major release. Incremental upgrades were possible when only the patch-version differed. During this development period, the communication protocol to submit jobs and get status information remained unchanged within the same major release. Maintaining, supporting and documenting the two releases required significant effort which was provided in part by leveraging Grid and experiment support systems, such as the OSG's iGOC [8]. In addition, we used meetings and mailing lists, documentation and collaboration tools such as a Wiki [7] and the Savannah project portal [6], to help reduce the complexity of the development and support tasks.

To improve future versions, the errors encountered running the production system with the different Capone versions, similarly described in [9], have been collected and classified. Most of the errors involve data movement and can be reduced using throttling or retries. Some errors, like Capone host interruptions or proxy problems, could be eliminated by providing mechanisms for maintain and recovery of persistent state information.

PRODUCTION ON GRID3 AND OSG

The ATLAS production system in Grid3 and OSG, as in other Grids worldwide, follows a common schema [3] involving a "supervisor" layer, Windmill, interacting with a central production database at CERN and one or more "executors" components, such as Capone, which manage all interactions with the underlying Grid infrastructure. Figure 1 shows the different steps of the job workflow in the new version of Capone and its interactions with the main elements of the production system. The central part of the figure represents the core Capone components and its checkpoints. At left, additional components of the ATLAS production system are shown as are the computing and storage elements of the OSG and shared servers. In the upper right section are other programs, such as components from the GriPhyN virtual data system (VDS [14]) and Condor-G [2], which execute on the same submit host as Capone. Also on the bottom right are the monitoring system elements on the submit host and on the remote servers. The solid arrows represent the execution of a job from its submission, by Windmill, to its "completed" state in Capone.

Mambelli et al. [9] contains a detailed description of all the steps involved in the job execution. Here we will focus on the interactions between components and on the improvements from the previous architecture.

*marco@hep.uchicago.edu, ** rwg@hep.uchicago.edu
#jerryg@anl.gov

The workflow starts with the job supervisor, Windmill, sending an XML message to Capone containing a job execution request. Before acknowledging the acceptance of the job, Capone checkpoints on disk all necessary job-specific information. From now on the job has been delegated to Capone for management and execution. The state persistence mechanisms introduced maintain knowledge of the job even after catastrophic events like a crash of the submit host, allowing the delegation to be fulfilled successfully at all times.

The next phase, detailed in [9], involves the interaction with some external servers and several executables running on the submit host. This phase has a low failure rate (since it is mostly local) but it is responsible for most of the load on the submit host because of the many executables involved require their own Java virtual machine. In case of failure the whole process can be repeated since it modifies only local files and a local database. At the end of the process a concrete DAG (direct acyclic graph) is ready to be submitted via Condor-G to OSG sites.

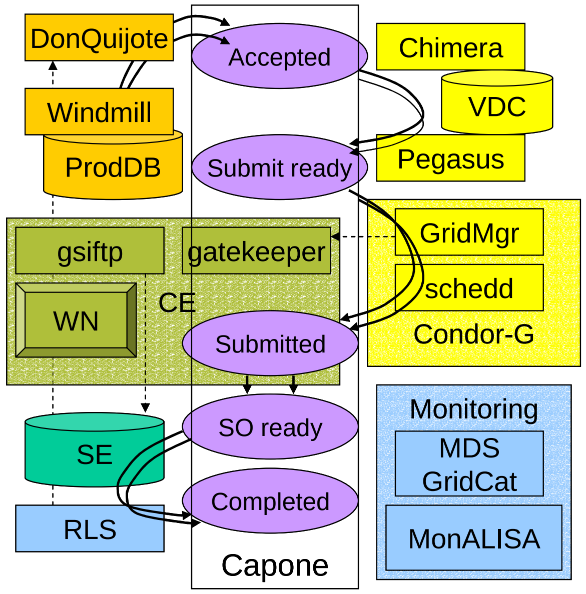

Figure 1: Job Workflow in Capone

Condor-G shadows each running DAG with a management process (DAGMan [2]) which insures proper execution of each node in the graph. In order to reduce the load on the submit host, DAGs are queued and grouped in batches before being submitted to a single DAGman process. When receiving new jobs, Capone queues them until the batch size is reached or a certain amount of time from the previous submission has lapsed; the user can configure both of these parameters.

Jobs submitted by Condor-G run on remote compute elements (CEs) and Capone checks through Condor to know the current state of a job and to find out when the job has completed. All the interactions with Condor are mediated by consumer processes to avoid overload. On the CE, if necessary, all input files are first staged-in, then the ATLAS software, an Athena execution, is started in a sandbox at the remote site under the direction of a VDS software executable named *kickstart*. A wrapper script, called by *kickstart* and specific to the transformation to be executed, is called first to ensure that the environment is set up correctly before starting any ATLAS-specific executables. The wrapper script also checks for any errors during execution and reports results back to the submitter through *kickstart*. In addition, it performs additional functions such as evaluating an MD5 checksum on all output files.

After the execution on the CE, Capone checks if the job completed successfully and starts the stage-out process, where the job output files are copied reliably (i.e. verifying file size and MD5 checksum) to their final destination. The transfers are performed using Globus gridftp. Since the gridftp servers, which in many cases are also GRAM gateway servers, are prone to overload and failure, Capone performs throttling by postponing the transfers until the number of transfers managed by both source and destination servers is below a certain threshold. The final step performed is the registration of output files and metadata in a file replica catalog. To improve the reliability of this step, registration failures are retried several times and Capone can handle also a list of alternative servers. It is easy to implement policies of active replication or backup. The file catalog in fact does not handle replication on its own, therefore if a final-stage replication is specified Capone takes care of it.

Interactions with the supervisor Windmill to report job status and to validate job execution and output file integrity are unchanged from the previous Capone version described in [9].

NEW ARCHITECTURE

To make Capone more resilient to failures and service interruptions in the grid infrastructure, several throttling and redundancy steps were incorporated into the code. These included file transfer *wait* and *retry* processes if the server was overloaded, and the use of a list of failover servers for events like file registration failures.

The main improvements on the submit host include job batching, the addition of (disk) persistency and crash recovery, checkpointing and backtracking, and a new finite state machine.

Batching is important to reduce the load on the submit host. Previously, each ATLAS job had its own independent submission workflow managed by a separate DAGman process. With batching, jobs are queued, grouped together and submitted as a batch to the Grid. This reduces the number of processes necessary for job submission and, more importantly, reduces the number of DAGman processes running on the submit host. There is a trade-off in choosing the right batch size; the bigger the batches, the more jobs a submit host will be able to handle, the smaller the batches the more responsive a system will be in submitting jobs as soon as they are received. In testing and debugging sites for Capone a batch size of 1 to 3 allows a quick turnaround but is unlikely to reliably manage more than 1000 jobs per submit host. A batch size of 10 or more was found to reliably scale job management to several thousands of jobs per submit host but with an associated slow down in the job submission rate.

Previous versions of Capone had a python thread following the state of each job. This approach simplified the programming because it replicated a process handling a single job and was reliable since a delay in a grid interaction or an unexpected exception affected only a job that might fail. More recent versions of python increase the amount of memory reserved for the thread stack. In machines with little memory or operating systems with tight memory management, the number of allowed processes could be as little as several hundred, thus limiting the maximum number of jobs that could be handled by Capone. The new approach is to have a few “special states” identified during the workflow where jobs are queued into a FIFO and multiple server threads move jobs from one of these to the following one. This way the total number of threads is limited and controlled by the configuration, but there are still multiple threads for each process allowing parallel processing and providing the same redundancy of the previous architecture. No single job can stop all the others because of a grid delay. No single job failure can affect the flow of other jobs.

The new architecture allows a straightforward implementation of state persistency management. Each job can be checkpointed before being added in the queues of the “special states”. The status saved is recoverable after a crash and together with Condor log files provides for a full recovery of all jobs (to resume almost from before the crash). It is also possible to save a snapshot of the current execution or to restore a previously saved one. This feature can be used to move all or some of the jobs to a different instance of Capone and to perform backtracking. If one or more jobs failed because of a transient error condition which has now been solved (e.g. a network interruption or an expired proxy) and the failure has not been communicated to the supervisor, these jobs can be moved back to their last healthy status and restarted from that point.

Other minor architecture changes included the use of a directory tree to store job information since the previous flat structure did not perform as well and could not hold more than 32000 jobs because of an ext3 file system limitation, and several utility scripts and programs to: rotate log files, compress or remove information about completed jobs, and provide additional help in debugging, troubleshooting, WS job submission and analysing the local log files and the remote CE execution directory.

BENCHMARKS

Several tests have been performed to evaluate the new software under normal operation and overload conditions. Each test consisted of starting Capone on a submit host, submitting jobs using *cclient* and measuring submit rates, load on the host, and number of jobs in different states.

Table 1: Test results per submit host for a typical Capone job management session (“C-G” indicates Condor-G managed jobs).

Measure	**Avg**	**min**	**max**
Submission rate (jobs/min)	541	281	1132
Grid Submit rate (jobs/min)	18	15	48
Running jobs	4019	0	6746
Total jobs	7176	0	8100
C-G running	385.4	68	596
C-G pending	345.1	14	537
C-G unsubmitted	564.9	0	1344
C-G total	1310.4	82	2481

The mix of test jobs included actual ATLAS jobs (short event generation and detector simulation jobs) and simple test jobs (CPU intensive and/or ‘sleep like’ jobs). Each test was repeated several times and the results collected in Table 1 are the average, maximum, and minimum values observed in different repetitions of the test.

Further tests verified the reaction of Capone to failures of external components of the system, like a crash of the machine or a network interruption.

Test 1: Scalability and Load Evaluation

A test to evaluate performance of a Capone job management session consisted of submitting jobs in a ramp-up phase until the number of jobs was about twice the number of available CPUs, and then maintaining that set of jobs. The submit rate, measured both during the ramp-up phase and during the steady state phase, was almost independent of the total number of jobs: a job that is running in a remote CE or staging in or out files causes little load on the Capone submit host. The submission rate was instead more sensitive to the number of jobs still in the submit state. Both decoding the XML message associated with job submission and executing the local steps from the VDS components are CPU intensive. The failure rate during this test was around 1.5% excluding the DNS failure mentioned below. While the CPU load on the submit host can reach values as high as 70 during job submission, the machine never became unresponsive, with the load returning to below 1 once all jobs completed the submission steps. The DNS failure during one of the iterations caused the failure of almost all the running jobs (due to timeouts) and provided a reason to test the recoverability of the jobs. All jobs recovered succesfully.

Test 2: Overload Conditions

Another test was used to test the system under heavy load conditions. The goal was to keep more than 4000 jobs running for an extended period of time (4-6 hours).

The results are similar to those of the previous test. The number of jobs actually running on the worker nodes of the CE remained around 500. This number is bounded by the available CPUs, while the jobs queued at the CEs or in the submit host can be higher. This increment causes higher load in all the components of the distributed system and a higher failure rate around 6.5%; mainly concentrated on a few overloaded CEs, without increasing the actual speed at which the jobs were executing and completing.

Test 3: Recoverability from System Crashes, Backups and Snapshots

After a catastrophic failure of the submit host, like a system crash, a shutdown, or after killing all Capone processes, Capone can be restarted in recovery mode and will pick up from where it left off. Most of the jobs will be able to continue without problems. The few observed failures were due mainly to timeouts in Condor or GRAM or the partial execution of external interactions that were not idempotent, like the registration in the file catalogue. The rollback to a previous checkpoint of the failed jobs, after some recovery action if required, allows recovery of most of these failures.

The rollback mechanism also proved to be a powerful solution to recover jobs after transient failures like the expiration of the proxy or a network failure. In this case Capone had to be stopped, the problem was diagnosed and removed, e.g. by renewing the proxy or reinstating the connectivity. Then some recovery action had to be performed like removing a partial transfer or a partial registration. Finally the job status can be reverted to a previous state and Capone restarted in recovery mode.

The recovery time is proportional to the number of jobs. For a submit rate of about 1000 jobs/min Capone took less than 7 min to recover more than 8000 jobs.

The recovery and rollback mechanisms included the possibility of taking a snapshot of the current status for backup purposes or for later re-execution. Using a backup, it is possible to have a different submit host continuing from where the original one left off. Another recovery procedure that worked during the tests, but is not recommended for production, is to select jobs from different snapshots and add them to a running or entirely new Capone instance.

RELATED WORK AND CONCLUSIONS

There are many software frameworks which have been developed for similar, but experiment-specific workflows in high energy physics. Others have been developed to handle generic applications on the Grid. DAGMan [2] requires a generic workflow specified as a sequence of nodes interconnected in an acyclic graph. It is flexible and reliable but somewhat heavy and difficult to use directly. Other solutions like the Lexor [10] or CG-executor [12] move part of the workflow in a script and rely on external components to hold the job state information. Dirac [11] implements a job "pull" model where a placeholder job request is made to a central server once a resource has been acquired. It uses resources less efficiently but is more responsive and provides a slightly different workflow, tailored to the needs of another LHC experiment. PanDA [4] is the new ATLAS job execution system for OSG and has a partially data-driven model similar to Dirac: brokerage places the data, a job dispatcher uses the brokered information and sends the job when data has been identified as having been transferred.

Capone is a flexible tool targeted to ATLAS production but able to execute different tasks such as analysis or general user scripts. The tests show that it can scale to more than 4000 jobs per submit host (far more than the currently available resources), recover completely from a host crash, and reduce or eliminate several errors experienced during DC2 and Rome production. Submission rate and responsiveness could be further improved by submitting multiple jobs at the simultaneously or by moving to a pull model such as developed in Dirac and PanDA.

ACKNOWLEDGEMENTS

This work was supported in part by the US Department of Energy and by National Science Foundation Grants ITR-0122557 and PHY/ITR-0113343. Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, under contract W-31-109-Eng-38.

REFERENCES

[1] A Toroidal LHC ApparatuS, http://atlas.web.cern.ch/
[2] DAGman: http://www.cs.wisc.edu/condor/dagman/
[3] K. De at al, "Lessons from ATLAS DC2 and Rome Production on Grid3", *CHEP*, 2006
[4] K. De et al. "Panda: Production and Distributed Analysis System for ATLAS", *CHEP*, 2006
[5] The Grid2003 Project "The Grid2003 Production Grid: Principles and Practice", *HPDC13*, 2004, Honol., HI, Grid2003, http://www.ivdgl.org/grid2003
[6] http://griddev.uchicago.edu/savannah/projects/atgce/
[7] https://uimon.cern.ch/twiki/bin/view/Atlas/Capone
[8] iGOC http://goc.ivdgl.org/
[9] M. Mambelli et al., "ATLAS Data Challenge production on GRID3", *CHEP*, 2004
[10] D. Rebatto "The LCG-2 Executor for the ATLAS DC2 Production System", *CHEP*, 2004
[11] A. Tsaregorodtsev et al. "DIRAC: A Scalable Lightweight Architecture for High Throughput Computing," grid, pp. 19-25, *Fifth IEEE/ACM Intl. Workshop on Grid Computing (GRID'04)*, 2004
[12] R. Walker, M. Vetterli, et al. "A Grid of Grids using Condor-G", *CHEP*, 2006
[13] F. Wuerthwein, R. Pordes, "The Open Science Grid", *CHEP*, 2006
[14] Y. Zhao, M. Wilde, et al. "Virtual Data Grid Middleware Services for Data-Intensive Science", *Middleware 2004*, August 2004, Toronto, Canada