Production Management System for AMS Computing Centres

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 092034 doi:10.1088/1742-6596/898/9/092034

Production Management System for AMS Computing Centres

V Choutko[1], O Demakov[1], A Egorov[1], A Eline[1], B S Shan[2,4] and R Shi[3]

[1] Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, USA
[2] Beihang University, 37 Xueyuan Road, Haidian Qu, Beijing 100191, China
[3] Southeast University, 2 Sipailou, Xuanwu Qu, Nanjing, Jiangsu 210018, China

E-mail: baosong.shan@cern.ch

Abstract. The Alpha Magnetic Spectrometer [1] (AMS) has collected over 95 billion cosmic ray events since it was installed on the International Space Station (ISS) on May 19, 2011. To cope with enormous flux of events, AMS uses 12 computing centers in Europe, Asia and North America, which have different hardware and software configurations. The centers are participating in data reconstruction, Monte-Carlo (MC) simulation [2] /Data and MC production/ as well as in physics analysis.

Data production management system has been developed to facilitate data and MC production tasks in AMS computing centers, including job acquiring, submitting, monitoring, transferring, and accounting. It was designed to be modularized, light-weighted, and easy-to-be-deployed. The system is based on Deterministic Finite Automaton [3] model, and implemented by script languages, Python and Perl, and the built-in sqlite3 database on Linux operating systems. Different batch management systems, file system storage, and transferring protocols are supported. The details of the integration with Open Science Grid are presented as well.

1. Introduction

Table 1 lists 12 computing centers. The centers have various hardware/software configurations, storage , and batch systems. For example, several batch systems, including LSF [4], HTCondor [5], SGE [6], Slurm [7] and LoadLeveler [8] are used by different sites. To facilitate the production management in the computing centers, a production management system has been designed and implemented.

[4] Corresponding author

Table 1. AMS Computing Centers

Centre	Location	Logical cores	Hardware type	File systems	Batch System
CERN	Geneva, Switzerland	6000	X86_64	EOS+AFS +CVMFS	LSF/HTCondor
CNAF	Bologna, Italy	1300	X86_64	CVMFS +GPFS	LSF
JUROPA /JUAMS	Juelich, Germany	4000	X86_64	LUSTRE +GPFS +CVMFS	Moab/TORQUE Slurm
MIT/OSG	Boston, US	Up to 6000	X86_64	CVMFS	HTCondor
SEU	Nanjing, China	2016	X86_64	GPFS	LSF
NLAA	Beijing, China	1024	IA64	CXFS	–
IHEP	Beijing, China	Up to 1500	X86_64	LUSTRE	LSF/HTCondor
IN2P3	Lyon, France	500	X86_64	GE	
Acad. Sinica	Taipei, Taiwan	3000	X86_64	CephFS +EOS +CVMFS	HTCondor
RWTH	Aachen, Germany	1640	X86_64	LUSTRE +CVMFS +AFS +FhGFS	LSF/HTCondor
JURECA	Juelich, Germany	Up to 20000	X86_64	GPFS	Slurm
JUQUEEN	Juelich, Germany	Test-only	PowerPC	GPFS	LoadLeveler

The central production management at CERN uses the Oracle Parallel Database [9] (PDBR) to store the production related information, and provides the job requesting and validating service for the computing centers. The job requesting service is provided by the Common Gateway Interface (CGI), querying the PDBR database about the availability of the production jobs, generating the job scripts, writing the metadata of the jobs into the PDBR database, and finally sending the job mail to the specific email address with the tar ball of the production job scripts attached. After the production jobs are finished at computing centers, the produced files, including the "journal" files containing the metadata, are transferred back to CERN, and the job validating service will check the consistency of the files, move the files to the final storage destination, and write the metadata into the PDBR database.

The production management system for the computing centers takes care of all the "intermediate" steps, from retrieving the job scripts from the job email, till transferring the produced files back to CERN.

By this way, the production management system for the computing centers is designed to run independently, which means it has a loose coupling (email as input, and produced files as output) with the central AMS production managing at CERN.

The production management system for the computing centers features:

- High-efficiency
  - To keep batch queue as full as possible and ensure efficient usage of computing resources;
  - To handle different types of production jobs, data and simulation, long lasting and flash jobs which may only run few minutes;
- Light-weighted
  - Not to require communication between processes;
  - Not to require super user privilege to deploy and (re)configure;
- Adaptable to different batch, storage and transferring systems
  - To be able to adapt various batch systems, storage systems, transferring methods, etc.;
- Customization
  - To separate site specific configurations/codes from general functional logic codes.

2. Deterministic finite automaton model

The goal is to have the highest possible efficiency for the production, i.e., to minimize the idle time of all working nodes. To achieve this, we design the system using the Deterministic Finite Automaton model, which ensures the computing resources are always filled by production jobs. The transition of the states is illustrated in Figure 1.

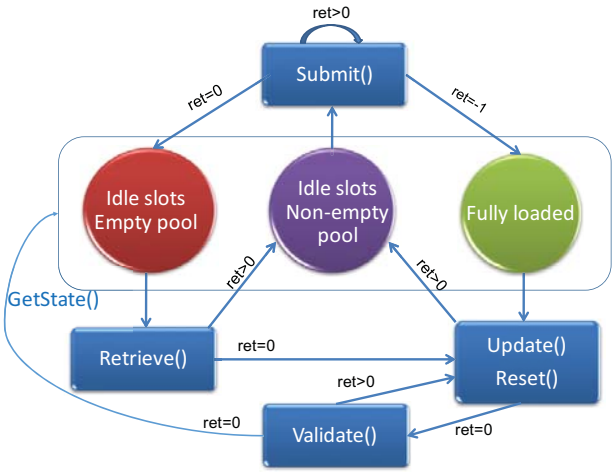

Figure 1. The production management system's design based on the Deterministic finite automaton model, "ret" is the return code of the corresponding function.

Depending on the status of the production farm and job pool, three states are defined as followed.

- Idle slots and empty job pool
  - There are free slots in the production farm and there is no job in production pool;
  - In this state, the system will retrieve production jobs;
  - If jobs are retrieved (i.e. return code `ret` is positive), the DFA will turn to the state "Idle slots and non-empty job pool";
  - Otherwise, the system will continue with updating job status and resetting failed jobs, then try to retrieve again;
- Idle slots and non-empty job pool

  - There are free slots in the production farm and there are jobs in production pool;
  - In this state, the system will repeat filling the free slots by submitting jobs;
  - If slots are all filled (i.e. function `Submit()` returns `-1`), the DFA will turn to the state "Fully loaded";
  - Otherwise, the DFA will turn back to the state "Idle slots and empty job pool";

- Fully loaded
  - All computing slots are filled in the production farm;
  - In this state, the system will update the status of submitted jobs, and reset those failed jobs;
  - If after updating/validating there are idle slots (i.e. return code `ret` is positive), the DFA will turn to the state "Idle slots and non-empty job pool";
  - Otherwise, the system will find those finished jobs to validate, and try updating/resetting again.

Figure 2. When starting a production campaign, the initial state of the DFA is "Idle slots and empty job pool", so the system begins with job retrieving. In the starting stage, the state remains in "Idle slots and non-empty job pool", and during the full speed running stage, the DFA state switches between "Fully loaded" and "Idle slots and non-empty job pool", until all the jobs in pool are submitted. After there is no more job to be retrieved or submitted, the production goes to the finishing stage, when the DFA state stays in "Idle slots and empty job pool", and the system repeats updating/resetting and validating.

Figure 2 shows how the three DFA states transfer during a production campaign. The main loop of the production management daemon starts from `GetState()` to get the current state, and goes through, and finally ends when `Validate()` finds nothing to be validated. By this design, it ensures a quick startup by continuous retrieving and submitting, a stable full speed running by updating/resetting and submitting, and also a fast finishing by frequent updating/resetting and validating.

Based on this DFA model, the production management system achieves the high efficient usage on resources. Figure 3 shows the queue status as a function of time when we reconstructed our science data at JUROPA (now renamed to JUAMS) computing centre in 2012. As mentioned in Table 1, there are 4000 physical cores with 2 way Hyper-Threading enabled which are shared by the 16-threaded production jobs and some analysis jobs. The production management system keeps a certain amount of jobs in the queuing status during the production period (except the hardware maintenance).

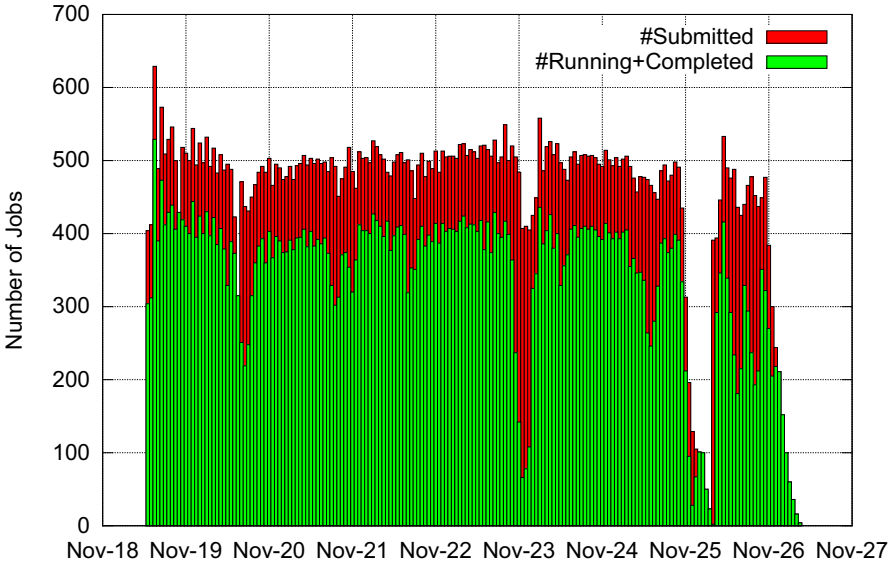

Figure 3. The queue status of the data reconstruction in 2012.

3. Light-weight design

A light-weight design is important to ensure a quick deployment in a new computing site or upgrading of an existing site. We use as much as commonly deployed tools and include all supporting libraries/software in our distribution package.

3.1. Programming languages

The production management system is written in Perl [10] and Python [11]. Most functions of the system are written in Perl, except that the transferring part is written in Python, because it requires multi-threading feature to improve the efficiency.

3.2. Database

All the information related to production jobs is stored in a database so that the whole production process can be tracked. SQLite3 [12] is selected as our database engine since it is included in most Linux distributions, and various programming interfaces are provided, including Perl and Python.

The database include 4 tables: DATAFILES to store the information of input raw files for data production and output raw files for simulation production; JOBS to store the information of all production jobs; NTUPLES to store the information of output ROOT [13] files for all production jobs, and JOBRUNS to store all the running information (the starting time and finishing for each production job, the finishing status, the execution site/host, the failing reason, etc.).

4. Customization

Different computing centers usually have different configurations on hardware, software, storage, etc. To adapt the variations, we peel the configuration related functions off the codes, and put them into a configuration file of environment variable definitions. This includes not only traditional "configurations", like the production home directory (`AMSPROD_HOME`), and the batch pool size (`AMSPROD_QUEUE_SIZE`), but also some production actions. For example, for job

submission, we define an environment variable `AMSPROD_CMD_SUBMIT`, where we can define the real command (template) of job submission. In this case we don't have to redesign the code when changing batch system from LSF to condor.

All the commands related to batch system and transferring tools have been defined in the form of `AMSPROD_CMD_`, with "variables" which will be replaced during runtime, like the file path to be transferred.

5. Diagnostics on resources

Open Science Grid [14] (OSG) provides facilitates access to distributed high throughput computing for research in the United States, and we started to run production jobs on OSG since February 2016.

As OSG has over 100 individual sites and a variety of resources, it is important to identify specific failing/problematic nodes/sites and exclude them from job execution. This system provides automatic diagnostics on log files: it records the statistics of each job execution in the database, including the host and site, the status, etc. In case a job execution fails, it will analyze the log file and record the failing reason(s) as well, and:

(i) Create a "blacklist" consisting of the recent "unreliable" sites;
(ii) Notify the administrator about the site-specific issues.

For example, in case all the jobs executed on site A in the past hour failed due to CVMFS issue, site A will be put into the blacklist in OSG submission script and a notification mail be be sent to the administrator.

6. Applications and performance

The described production system has been deployed in JUROPA/JUAMS Germany, CNAF Italy, IN2P3 France, IHEP China, CERN, and MIT/OSG USA. These sites made major contributions on AMS data production, and are also active in Monte-Carlo simulation.

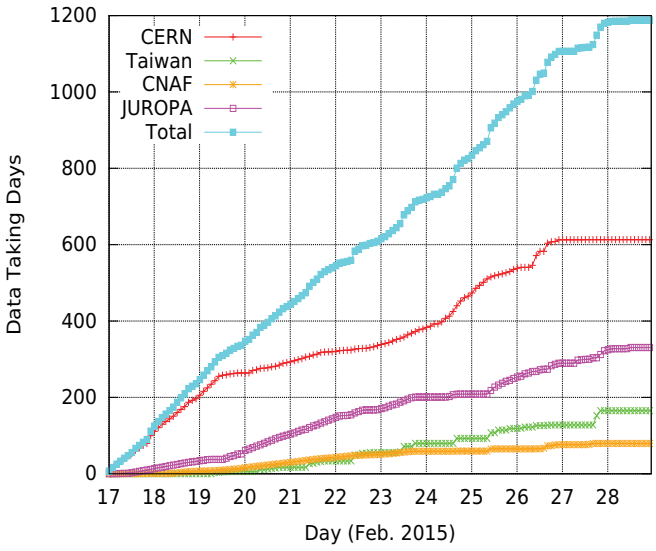

Figure 4. The completion status of the science data reconstruction in 2015.

Figure 4 shows the completion status of the science data reconstruction in 2015. Totally 42 months' data, 54 billion events, were reconstructed in 12 days, and the reconstruction speed reached more than 100 data collection days per production day.

Figure 5 shows the completion status of the Helium Monte-Carlo simulation campaign at the end of 2015 and the beginning of 2016. In around 3 months, more than 5536 years' CPU time was delivered, 2.24 trillion triggers were simulated, and 35.5 billion events were recorded, which resulted in nearly 1 PB data.

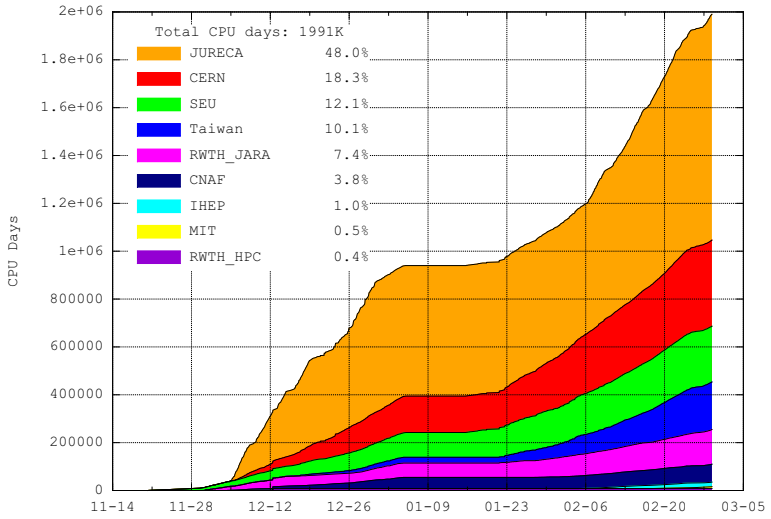

Figure 5. The completion status of the Helium MC simulation in 2016.

7. Conclusion

The production management system for computing centers is designed using the deterministic finite automaton which ensures the computing resources are always filled by production jobs. It is implemented by script languages and Linux built-in database, which makes the deployment and customization fast and easy.

The recent development of the system includes the integration of OSG resources, and automatic diagnostics on OSG sites and hosts.

The system has been deployed in most of AMS computing centers and used for daily production management.

References

[1] Ting S 2013 *Nuclear Physics B-Proceedings Supplements* **243** 12–24
[2] Binder K 1987 *Quantum Monte Carlo Methods* 241
[3] Cohen D I and Chibnik M 1991 *Introduction to Computer Theory 2nd edition* (Wiley)
[4] Zhou S 1992 LSF: Load sharing in large heterogeneous distributed systems *I Workshop on Cluster Computing* vol 136
[5] *http://research.cs.wisc.edu/htcondor/htc.html*
[6] Gentzsch W 2001 Sun Grid Engine: Towards creating a compute power grid *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on* (IEEE) pp 35–36
[7] Yoo A B, Jette M A and Grondona M 2003 Slurm: Simple linux utility for resource management *Workshop on Job Scheduling Strategies for Parallel Processing* (Springer) pp 44–60
[8] Prenneis Jr A 1996 *Proceedings of Supercomputing Europe (SUPEUR)* **176**
[9] DeWitt D and Gray J 1992 *Communications of the ACM* **35** 85–98
[10] Wall L *et al.* 1994 The perl programming language

[11] Van Rossum G *et al.* 2007 Python programming language. *USENIX Annual Technical Conference* vol 41
[12] Owens M and Allen G 2010 *SQLite* (Springer)
[13] Antcheva I, Ballintijn M, Bellenot B, Biskup M, Brun R, Buncic N, Canal P, Casadei D, Couet O, Fine V *et al.* 2011 *Computer Physics Communications* **182** 1384–1385
[14] Pordes R, Petravick D, Kramer B, Olson D, Livny M, Roy A, Avery P, Blackburn K, Wenaus T, Würthwein F *et al.* 2007 The open science grid *Journal of Physics: Conference Series* vol 78 (IOP Publishing) p 012057