Review of existing Monte Carlo production mechanisms in ATLAS and prototype distributed production systems

C. L. Tan
School of Physics and Astronomy
University of Birmingham, UK

February 18, 2003

Abstract

This review attempts to provide a comprehensive though not exhaustive review of where things in the distributed production areas are headed. It is a resource to guide new developments in the abovementioned areas by providing a better picture of what is out there. The first section of this report presents a review of the existing and developmental Monte Carlo production interfaces and mechanisms in ATLAS. AtCom, EDG GUI, GANGA, GENIUS and Grappa will be discussed. The next section reviews the various prototype distributed production systems across seven different experiments. These include AliEn(ALICE), Centipede(H1), Funnel(ZEUS), SP4(BaBar), Production Tool Suite(CMS), Request System(D0) and SLICE(LHCb).
Contents

1 Current ATLAS production mechanisms 3

2 ATLAS production mechanisms in development 3
   2.1 ATLAS Commander (AtCom) ........................................... 3
   2.2 EDG GUI ............................................................... 3
   2.3 GANGA ............................................................... 3
   2.4 GENIUS .............................................................. 6
   2.5 Grappa (US-ATLAS) .................................................. 8

3 Other distributed production mechanisms 13
   3.1 AliEn (ALICE) ......................................................... 13
   3.2 Centipede (H1) ....................................................... 15
   3.3 Funnel (ZEUS) ....................................................... 17
   3.4 SP4 (BaBar) ........................................................ 18
   3.5 Production Tool Suite (CMS) ................................. 23
   3.6 Request System (DØ) ............................................. 26
   3.7 SLICE (LHCb) ....................................................... 29

4 Conclusion 31

5 Acknowledgements 32
1 Current ATLAS production mechanisms

ATLAS Monte Carlo production, as of Data Challenge 1 Phase 2, has only just commenced large-scale distributed production. Although automated shell scripts are used throughout the production process, the distribution of work (i.e. the utilisation of computing power and storage servers) is still a manual process (i.e. pre-assigned and agreed upon by human co-ordinators). With Data Challenge 1 Phase 2, we will also begin to see Monte Carlo production running in parallel on the GRID using a combination of EDG middleware and Globus tools. These first few steps towards a truly distributed production system will allow ATLAS to deal with the anticipated multi-petabytes/year data rate when it starts to collect data in 2007.

Current Monte Carlo production using the GRID still requires the user to issue a considerable number of commands. Unfortunately, there is no official utility that will permit a non-expert user to configure a physics job, find input files, submit the job(s), monitor its status, retrieve the output, etc., all in a single environment/interface. However, there is considerable effort currently being undertaken to do just that. These mechanisms will be reviewed in the next section. In section 3, distributed production mechanisms from other experiments will be reviewed.

2 ATLAS production mechanisms in development

All production mechanisms described below are currently in different stages of development. They are GRID-oriented or GRID-aware and provide convenient GUIs and/or web-interfaces.

2.1 ATLAS Commander (AtCom)

AtCom is a Java-based graphical interactive tool currently in development for use in the ATLAS DC 1 Phase 2 production. AtCom allows the user to select the datasets to use, define partitions and number of jobs required, select the transformations desired, choose the target structures, select the operations and specify the remote location to store the output. The user will be able to monitor the status of the production from the same GUI in AtCom. Interface to other software components (e.g. Magda, EDG middleware, etc.) is possible by means of Java plugins.

AtCom is a very portable tool that will work under most operating systems. A working prototype is being tested and being prepared for deployment in the official pile-up production.

2.2 EDG GUI

There was insufficient information regarding the EDG GUI to provide a useful review. However, Figure 1 shows a montage of several screenshots of the prototype EDG GUI.

2.3 GANNA

GANNA, Gaudi/Athena aNd Grid Alliance, is a joint effort between ATLAS and LHCb that aims to simplify management of analysis and production jobs for end-users by providing tools for accessing GRID services with built-in knowledge of how Gaudi/Athena works.
Figure 1: Screenshots of the prototype EDG GUI.

Figure 2: GANGA’s architectural overview.

GANGA will provide the user with a single consistent desktop interface accessible from a local application or a thin client (remote user) for potentially all production and analysis tasks. Technical details of the GRID are hidden from the user: resource and data management will be convenient
and indistinguishable from local ones. The idea is that GANCA will be functionality analogous to a email system, with jobs as the emails. GANCA intends to make configuring a Gaudi/Athena job and running it on the GRID as easy as sending an email!

The following describes what happens automatically when a user decides to run a job:

i) The workspace database is queried to obtain the required working environment (i.e. list of files, tools state, etc.).

ii) The job configuration database is accessed to obtain a particular (common) job configuration. Manual customisation is also possible, much like AtCom.

iii) Logical Filenames (LFN) of input data are resolved using the bookkeeping database.

iv) Software requirements of the job are determined.

v) GANCA determines if the job can indeed be run by the user by checking his/her experiment privileges, CPU and storage quotas, credentials, resource requirements and availability, network load, etc. Large jobs may be split up to be executed in an optimal fashion.

vi) GANCA then creates the Job Description Language (JDL) file(s) required for submission to the GRID.

Figure 3: The GANCA prototype GUI.

Once a job has been submitted and has commenced execution, the user is able to monitor the internal state of the program (e.g. event and error counters, plot histograms using ROOT and access log messages for the job). See Figure 3.
Finally, upon completion, GAN Ga will reassemble the job output (if previously split) and transfer it to the user specified location before updating the relevant bookkeeping databases.

Software components in GAN Ga will be based on the concept of a Python bus (see Figure 4) where different modules required to provide full functionality of the interface are glued\(^1\) together using Python (see Figure 4). Additional components can be easily added (e.g. MathLab, Mathematica) by means of a Python interface to the new component’s API.

![Python bus design](image)

Figure 4: Python bus design.

**Strengths**

Python, as the choice of programming language, is well-supported. With powerful scripting capabilities and a wealth of open-source libraries to tap on, rapid development of GAN Ga with easy maintenance and subsequent upgrades will be possible.

### 2.4 GENIUS

**Technology and architecture adopted**

Short for Grid Enabled web eNvironment for site Independent User job Submission, GENIUS, is a generic GRID-based web portal\(^2\) (powered by EnginFrame\(^3\) and embedded with EDG middleware and Globus tools). See Figures 5 and 6.

The various services provided by GENIUS are:

\(^1\)Python will provide the mechanism for components to communicate and interact with one another.

\(^2\)https://genius.ct.infn.it

\(^3\)EnginFrame is a flexible, scalable and highly customisable framework designed to provide an intuitive web-based interface for browsing compute resources whilst imposing policies on their access patterns. Depending on the client, EnginFrame converts XML-based resources to other more suitable formats (e.g. HTML, WML, PDF, etc.).
File services
This is a set of standard operations (e.g. create, view, edit, rename, delete, create directory, upload TARball, upload file and display environment variables) a user may perform on files stored on the remote UI machine.

Security services
*GENIUS* allows for password and certificate maintenance. Common commands like certificate upload, upload of `.globus` TARball, upload of `.p12` certificate, proxy info, renew proxy, password change and X.509 PEM phase change are provided.

Job services
Here, *GENIUS* allows the user to submit a job\(^4\) to the GRID, monitor the job status and retrieve the output.

The user prepares the JDL file, chooses a suitable *Resource Broker (RB)* and queries it to find all available *Computing Elements (CE)* for the job. Subsequently, the user may monitor the status of the job as it is being processed and clean\(^5\) the job queue if required.

\(^4\)Only single jobs are permitted in *GENIUS* v1.3. This limitation will be removed in future releases.

\(^5\)All jobs will be removed. Selective cleaning not implemented yet.
• **Info services**
  The user may browse the testbed for GRID resources he/she desires to use. Currently CERN, CNAF, MSU, WORLDGRID and GLUE testbeds are supported.

• **Monitoring services**
  There are links to various mapcenters e.g. DataGrid, TopoGrid and WorldGrid. In addition, **GENIUS** will report on a particular site the total CPU used, the disk space available, current running jobs and those that are currently waiting.

• **Interactive services**
  **Virtual Network Computing (VNC)** is used to export the UI’s desktop session to the user’s desktop. This is effectively a secure telnet session with the UI within the **GENIUS**’s web interface.

• **Virtual Organisation (VO) services**
  As **GENIUS** recognises certificates from many VO’s. It allows a user to browse personal certificates of other users within the same VO. Users can also lookup logical file names (LFNs) in the VO’s **Replica Catalogue (RC)** and retrieve the corresponding physical file names (PFN).

### 2.5 Grappa (US-ATLAS)

**Technology and architecture adopted**

**Grappa** is an example of a **GRID Science Portal** which allows the user to work within an application/experiment specific environment and submit/monitor/manage jobs on the GRID via a web-based interface. It is based on Indiana Universitisy’s **XCAT Science Portal Technology** which uses tools like **https, Java, Globus, HTML** and **Jython** scripting.

The **XCAT Science Portal** includes the following components:
A notebook database. From the web interface, standard commands to create, open, delete, modify, import and export notebooks are provided. There are four default notebooks provided by Grappa: two contain information on using notebooks, the third contains a sample script that executes a command on the machine where the Portal web server is run, and the fourth submits an Athena job to the US-ATLAS testbed. Grappa utilizes a weighted round-robin selection technique for compute resources at each particular site. Using an application notebook, Grappa allows a user to submit and execute, in parallel, pre-constructed scripts at pre-determined locations.

- A script engine which allows sets of instructions to be executed on the GRID. Jython, a very flexible scripting language, is used here to access Java classes e.g. Globus Java CoG kit, XCAT and XMESSAGES.

- An event subsystem which deals with messages arriving from the GRID or from user applications.

- A GRID performance monitor provides information on the status of GRID resources.

- A remote-file management system which allows secure file transfers.

![Grappa/XCAT Science Portal architecture](image)

Figure 8: Grappa/XCAT Science Portal architecture.

Hence, the Grappa prototype is essentially a XCAT Science Portal with an Athena notebook\footnote{A Notebook is essentially a self-contained workspace that collects together files (documents, executables, scripts, etc.) relating to a specific application.} that allows Athena-based simulation jobs to be run on the US-ATLAS Testbed. It provides the user with visual access to GRID resources (i.e. compute elements and the MAGDA catalogue) and both visual and script-based frameworks for job submission (see Figure 9).

A typical Grappa user session\footnote{Refer to http://gate.hep.anl.gov/gfg/grappa/athena/ for a step-by-step introduction to using Grappa.}

The user brings up a web browser and invokes the portal. He/she selects the appropriate testbed resources and proceeds to configure the input files. Once that is done, the job may be submitted (see Figure 10). Monitoring is usually carried out at this stage: the user opens the monitoring window to ensure that the job runs to completion without any errors (see Figure 11).
Work in development

Grappa’s developers have noted that although the current version performs as specified, extending its functionality will not be easy. A rethink of Grappa’s architecture is required (see Figure 12). In the next version of Grappa, the XCAT Science Portal will be replaced by a modular Portal framework (e.g. Apache Jetspeed). This allows for the separation of visual appearance (multi-user customisation) of the web interface and its functionality packaged into portlets (not very different from Java plugins).

The way is paved for Grappa to evolve into Virtual Data Browser (VDB) which is very much like a web browser/composer. The VDB queries the Virtual Data Catalogue (VDC) and enables the user to search and publish virtual data. It will also provide customisable user profile management facilities (e.g. bookmarks, credential mgmt, resource mgmt, etc.).

More US-ATLAS production matters

Apart from Grappa, the other main software components used on the US-ATLAS GRID testbed for the Athena-Atlfast production are (see Figure 13):
Figure 10: Generic job submission using Grappa to the GRID.

Figure 11: Monitoring the running job status.

- GRID Applications Toolkit (GRAT)
  
  GRAT is a collection of more than 50 command-line shell and Python scripts using facilities
Figure 12: A comparison of the current Grappa architecture and the next release.

provided by Globus, Magda and MySQL to perform Athena-Alfast production. Its modular design allows for individual parts to be run independently. GRAT serves as a rapid development and diagnostics tool to test GRID middleware and their performance.

- Manager for GRID Data (Magda)
  At the core of Magda is a MySQL database, but the bulk of the system is in a surrounding infrastructure for setting up and managing (1) distributed sites with associated data locations, (2) the data store locations within those sites, and (3) the hosts on which data-gathering servers and user applications run.
  Typically, Magda gathers data from the various different types of data stores, allows users to manage the catalogue information via web interfaces and provides an API to allow production and end-user applications to replicate and serve files.

- Virtual Data Catalogue (VDC)
  With the introduction of VDC, the importance of production recipes have been raised to match and even surpass that of created data. Previously lost or discarded, production recipes are now stored as virtual data in the VDC. In fact, production recipes are now of primary importance and data of secondary importance as recipes can reproduce data. From the OO perspective, recipes are encapsulated together with corresponding data to create Virtual Data Objects.
  The features of a VDC-based production system are:

    - High-throughput
      This is made possible by the scatter-gather data processing architecture.

    - Fault tolerance of agents/jobs
      A combination of measures including the decoupling of agents, local caching of input and output data and pull-model task assignments provided the much needed redundancy and resilience.

    - Automatic garbage collection by the job scheduler: once an agent pulls a derivation from the VDC and the data successfully materialised, the agent registers success in the VDC. Unsuccessful invocations (usually due to timeouts) are invoked again.
3 Other distributed production mechanisms

3.1 AliEn (ALICE)

Technology and architecture adopted

AliEn is a GRID prototype which is written in Perl5 and uses the Service Oriented Access Protocol (SOAP) to communicate and bind components together. AliEn manages the production of events centrally whilst it distributes processes and stores events across different sites.

AliEn is made up of the following components:

- Distributed Catalogue (DC)
The primary function is to map each logical filename (LFN) to one or more physical filenames (PFN) and it also stores Meta data. The AliEn catalogue resembles a standard UNIX file structure with a tree structure and different user and file privileges.

Its command-line interface is very similar to that of a UNIX shell with all the familiar set of commands. An additional set of functions allow for file catalogue entry maintenance e.g. add, remove, update, etc.

To improve on usability, the AliEn file catalogue comes with a GUI as well. See Figure 14.

- Authentication Service (AS)
A user client connects to the AS using its AFS password that is verified locally. A new user will have an account created and they are issued with a token encrypted with a secret key. A proxy server (i.e. the AS) recognises the user client by means of this token which has a limited validity of 24 hrs. With this token, the real password is not required.

- Queue Server (QS)
This is the central server which manages all jobs that have been submitted for execution. It accepts job submissions from any CE, dispatches the job to a suitable CE for execution and provides status of the job execution.
• Computing Element (CE)
The CE is a cluster of computers that are used for processing jobs. The CE obtains jobs from the QS and sends them to the local batch system.

• Storage Element (SE)
This is simply an interface to a local Mass Storage System (MSS).

• Information Server (IS)
The IS keeps track of SEs that are up and running. It is also queried whenever there is a file transfer request.

• Logger Service
This is a central log server which accepts and compiles all errors and warnings from the entire AliEn system.

Figure 14: The AliEn distributed file catalogue.

Production workflow

Assuming all certificates have been created and are valid, ALICE Monte Carlo production is simple and straightforward:

i) Locate files
Run the AliEn System by issuing the alien command at the UNIX prompt. This loads the shell-like interface of AliEn. Locate the files (i.e. LFN) required for the job with standard ls command.
3 OTHER DISTRIBUTED PRODUCTION MECHANISMS

3.2 Centipede (H1)

Centipede is a package of Fortran programs and shell scripts designed primarily for H1 Monte Carlo production.

Figure 15: AliEn’s web interface.

ii) Job submission
To submit a job to AliEn, type alien login at the UNIX prompt. This again brings up the shell-like interface but with job submission commands enabled. A JDL is required to describe each job. Its format and details can be found in the AliEn manual. Issue the submit command with the JDL file as input. The status of the submitted job can be queried through a webpage\(^8\) or by using the top command.

iii) Output retrieval
The output of any job is sent to stdout by default unless otherwise specified in the JDL.

GRID compatibility

Although the AliEn System is a GRID-like system, it is not supposed to be a replacement for GRID in the long term. The plan is to gradually incorporate EDG middleware as it becomes available and mature.

---

\(^8\)http://alien.cern.ch/Alien/DisplayQueue
Technology and architecture adopted

*Centipede* is heavily influenced in terms of functionality by the *Funnel* package (see Chapter 3.3) of the *ZEUS* experiment. Like *Funnel*, *Centipede* is designed to utilise spare cpu cycles on idle UNIX workstations distributed over a number of sites with minimum impact on them. *Centipede* uses the *PVM* package to control the spawning and monitoring of tasks and the passing of messages and data between tasks.

There are three main software components in *Centipede*:

- **Master control**
  *Centipede* is a master/slave system. The *master control* program runs at a central site where large data files are usually stored whilst slave tasks are launched on other nodes under the control of the master to perform computation.

  The *master control* program keeps track of the flow of events between tasks, schedules node usage, restarts crashed tasks and provides a logging service for monitoring purposes.

- **Master / PVM / User I/O package interface**
  This, as its name suggests, enables components to fit and communicate with one another.

- **Job descriptors and task scripts**
  Each *Centipede* job requires a host configuration file and a task configuration file. The host configuration file (*cpdhosts*) contains a list of the full addresses of all the hosts the job is allowed to use whilst the task configuration file defines the execution sequence and parallel configuration of the tasks to be run.

  Eight *H1* task scripts are supplied with *Centipede*. These scripts can be combined in any sequence the user deems fit depending on the result desired.

All job scripts invoke a common setup and cleanup script at the beginning and end of the task. The setup script (*prologue*) unpacks keyword arguments into variables, checks that all variables have been initialised, creates the working directory and write a short introductory blurb to *stdout*. The cleanup script (*epilogue*) deciphers the exit status codes of the executable, provides a stack track if a core dump occurred, returns (by e-mail or file transfer) all *stdout* and log files to the master host before deleting the working directory.

A job is executed using the *cpdrun* command with relevant input parameters for a particular run. PVM is started automatically.

**Strengths**

- The system has proven to be very reliable: crashed tasks are automatically restarted and lost events automatically rerun.

- Very easy to use. Only a few virtual jobs needed instead of hundreds of standalone jobs.

- Jobs may run on machines with no batch system and schedule policy easily implemented.

- No event data needs to be stored on slave nodes.
3 OTHER DISTRIBUTED PRODUCTION MECHANISMS 3.3 Funnel (ZEUS)

Current issues

- Constant network connection between master and slave nodes required. i.e. Bandwidth usage is considerable.
- Firewalls pose a problem.
- The master node is the single point of failure. When it goes down, the entire system grids to a halt. The whole virtual job is lost as well.
- Event ordering is not preserved as they are output as they are received from each slave task.
- Strictly speaking, jobs cannot be repeated as it is not known which tasks will process which events.

GRID compatibility

Although not designed to be GRID-aware, the Centipede system should easily interface with a Computing Element (CE) in a similar fashion as it would to PBS.

3.3 Funnel (ZEUS)

The Funnel software package is the ZEUS collaboration’s Monte Carlo event production system. Extensive automation and fault tolerance was the key to its successful adoption.

Technology and architecture adopted

The ZEUS Monte Carlo production mechanism is centrally controlled and gets its processing power from remote computers around the globe by harnessing idle CPU cycles that would otherwise be wasted. Configured at the lowest priority, Funnel clients do not interfere or compete with real users for resources. The Funnel program achieves levels of distributed parallel computing power equivalent to approximately one hundred dedicated computers for ZEUS’s Monte Carlo production needs. Figure 16 shows independent events distributed across many copies of the processing program by the para-manager program.

The main tools provided by the Funnel package deal with job queues, execution and failure of the processing program, parallel processing, data buffering, archiving and remote transfer. As Centipede (see Chapter 3.2) is based on Funnel, it is not difficult to see similarities in their design. The most obvious similarity is that jobs to be run are placed in a local transfer directory. Funnel maintains a daemon process that checks if there are jobs in this directory to be run. All jobs found are executed and their output archived. In addition logs of every actions taken and exceptions handled are kept. Successfully completed input jobs are simply deleted or moved from the local transfer directory.

Production workflow

Production steps are few and simple:
3 OTHER DISTRIBUTED PRODUCTION MECHANISMS

3.4 SP4 (BaBar)

Technology and architecture adopted

The Monte Carlo production mechanism involves the following software components:

- ProdTools
  This is a suite of tools used by the production manager to initiate the simulation environment, maintain centralised control at SLAC and monitor simulation production to ensure that certain rules are adhered to. These tools attempt to provide an abstraction layer that shields the production manager from myriad production details.

- "Bob’s SLAC scripts"
  These shell scripts complement the abovementioned ProdTools by automating common maintenance, monitoring, logging, statistics collection and other frequently executed tasks. This effectively reduces the time required for simulation jobs.

- Monolithic Object-Oriented Simulation Executable (MOOSE)
  Figure 17 shows the architectural overview of BaBar’s event simulation system based on three executables, BgsApp (Bogus), SimAppApp (SimApp) and BearApp (Bear). These applications, though standalone executables, need to be run in strict succession. This is highly inefficient because intermediate data has to be retrieved from and stored to an Objectivity database between stages. By combining these three executables into a single executable, the number of database transactions required is greatly reduced and this translates into a significant improvement in the speed of the simulation.

Figure 16: Funnel architecture.

i) The user submits jobs to Funnel. Each of these jobs provides the input events (i.e. particle momentum vectors) required by the specified Monte Carlo simulation.

ii) For each job submitted, Funnel produces archived output events, log files and information on any input events that might have caused the simulation to fail.
MOOSE is the result of the amalgamation of these executables. In MOOSE, the intermediate data are stored internally (i.e., in memory) between stages and only the final event candidates are written out to the database on disk. MOOSE is currently being tested for simulation production and is expected to be deployed for SP5.

- **MocaEspresso**
  MocaEspresso is a tool that automatically exports Objectivity databases created during Monte Carlo production. It is capable of performing incremental extraction of the relevant files from the Objectivity database and transferring them to SLAC in parallel and asynchronously whilst the production is running.

- **BdbServer++**
  BdbServer++, a completely new and architecturally different implementation of the old BdbServer, is a web-based utility based on EDG/Globus file transfer tools. It allows a user at a remote site to make a copy of a collection of Objectivity data, extract it and transfer it back to storage locally. See Figure 18.

The web interface provides basic infrastructure for authentication and it is based on a single system for proxy delegation. More importantly, it is (1) a one stop site for a variety of packages / functions, (2) highly extensible making it easy to add extra facilities and (3) GRID-enabled (jobs have been successfully submitted to the Imperial College EDG Resource Broker). This utility is also capable of submitting Collib jobs (discussed later) to SLAC through the Globus interface. Web interfaces for Collib, SkimData and BdbCopyJob⁹ are currently in development and are near completion.

- **Data Catalogs**
  There is currently duplication in effort with respect to data catalogs but this is being reviewed. The main catalogs involved are:

  - Collection Database (CollDb)
    CollDb contains all collections of the analysis federation and SP1 analysis federation. Searching tools allow to search for collection names by run number(s), group, production release number, skim release number (for skim collections), mixing and/or reconstruction

⁹ BdbCopyJob is not discussed in this review due to a lack of information on it.
release number (for simulation federation). For each selected collection, information about associated *Objectivity* database files and their location can be obtained.

- skimData Tables
  These tables are very similar to *CollDbs* and are used by users to find out what *BaBar* collection names exist, where they are, number of events involved, etc. These tables contain *Objectivity* and *Kanga* information.

**Current issues**

It is suggested that input settings for the simulation and event data output be recorded in flat files for simplicity, especially when working in a GRID environment.

**GRID compatibility**

The *BaBar* Grid project is *BaBar*’s effort to exploit GRID technologies. Currently in a relatively early development phase, *BaBar* Grid is built predominantly on *EDG* middleware and some *BaBar*-specific software components. These *BaBar*-specific software components are:

- skimData Tables / Replica Catalogue
  With the recent addition of additional information about data at other sites (apart from SLAC), these *skimData* tables begin to resemble a replica catalogue. These *skimData* tables are mirrored (synchronised daily) at all sites and every site is aware of the data available everywhere else.

  A data query comprises of two steps:

  i) A user queries the database with *skimData* which returns all successfully matched local and remote entries. A *tcl* file is produced.
ii) This tcl file is fed into another script (bbmkidx) that checks the selected collections against the replica tables and generates a set of sequentially numbered tcl files containing collections that might originate from more than one site.

- New Collection Database (NCollDb)
  *NCollDb* contains various tables with information about databases (e.g. name, size, sweep, FDB, etc.) and collections (e.g. name, run, status, database, master/slave FDB, etc.)

- Storage Resource Broker (SRB) and Metadata Catalogue (MCAT)
  *SRB* is used for data distribution and entry search between sites. It is a client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets.

![SRB/MCAT setup at BBar.](image)

*MCAT* is a meta data repository with a uniform interface that is capable of storing and querying meta data. *MCAT*’s auto discovery mechanism can identify data of interest using a combination of characteristic attributes other than simply physical names and/or locations.

*SRB*, in conjunction with *MCAT*, provides a way to access data sets and resources based on their attributes rather than their names or physical locations.

The prototype *BBarGrid* job submission/retrieval web interface\(^\text{10}\) provides us a glimpse of how *BBar* will be performing production over the GRID. Although rudimentary, as with most prototypes, the web interface demonstrates that *BBar* has taken the effective steps towards the gridification of its simulation production. Plans for a fully featured GRID web-interface are on the way.

A typical web-based job submission sequence\(^\text{11}\) involves the specification of certain parameters like database type (i.e. *Kanga* or *Objectivity*), production nature (i.e. Real data, *Geant3* MC or *Geant4* MC), no. of events, no. of jobs, MC channel to use, event stream and the site to run the job.

\(^{10}\)http://www.slac.stanford.edu/babar-internal/spreq/requests.html

\(^{11}\)Screenshots of the web-interface from job submission through to retrieval of analysis results can be found at http://www.hep.man.ac.uk/groups/slach/gridinstructions.html
Figure 20: \texttt{BaBar} web-based GRID interface for job submission and retrieval.

Once submitted, the job status can be monitored. Upon completion, the job output is redirected for analysis. Output analysis histograms are presented back to the user through the web interface. See Figure 21.

Figure 21: \texttt{BaBar} output analysis histogram.
3.5 Production Tool Suite (CMS)

Technology and architecture adopted

![Diagram of CMS production tool suite](image)

Figure 22: The CMS production tool suite.

The main software components in the CMS production process as of the Spring02 Production were:

- RefDB (Central Reference Database)
  RefDB has two constituent SQL databases: the Central Input Parameter Database and the Central Output Metadata Database (see Figure 22). Through a secure web interface\(^{12}\) by means of HTML forms (see Figure 23), physicists register their production requests to the Central Input Parameter Database which records all process details (e.g. data name, application version, executable name and list of input parameters). These requests are subsequently assigned by the production coordinator of a particular Regional Centre (RC) who then returns (via e-mail) the corresponding AssignmentID. This AssignmentID then becomes the input parameter to IMPALA (to be discussed next).

Another web interface\(^{13}\) allows physicists to browse for the location of data and metadata by accessing the Central Output Metadata Database (see Figure 24).

- IMPALA (Intelligent Monte Carlo Production Local Actuator)
  IMPALA is the automated Monte Carlo production script generation tool. It takes the AssignmentID provided by RefDB and queries the latter for the corresponding request parameters, splits the job up into optimal chunks, creates the relevant Monte Carlo job scripts and submits them to a job scheduler (e.g. PBS). IMPALA then tracks the jobs through to completion.

- BOSS
  This tool performs job monitoring and book-keeping. As job schedulers are found in most computer farms, instead of re-implementing them, BOSS interfaces directly with them.

\(^{12}\)https://cmsdoc.cern.ch/cms/production/www/cgi/SQL/protec/mc_request.php

\(^{13}\)http://cmsdoc.cern.ch/cms/production/www/cgi/SQL/List_Datasets_for_browsing.php
Figure 23: CMS Production Request webpage.

Figure 24: CMS Metadata catalogue webpage.

- **DAR**

  DAR is simply a software distribution tool that allows the creation and installation of binaries. It is used predominantly to distribute the physics executables and geometry files.
• "Tony’s scripts"

Although CMS’s suite of production tools are predominantly automated, the distribution of assignments across the various Regional Centres (RC) is currently done by hand. Tony’s scripts are used by CMS to transfer data between RCs. This involves maintaining a simple HTTP server that publishes data files to clients who then request the server to push files of interest back to them using a variety of methods (e.g. scp, bscp and rfc).  

Production workflow

The following are the steps taken by CMS production staff at a RC for each assignment:

i) Staff receives assignmentID via e-mail and invokes the IMPALA ‘Declare’ script which then creates a list of parameters.

ii) These parameters are manually checked.

iii) The IMPALA ‘Create’ script is run.

iv) This is followed by the ‘Run’ scripts.

v) Staff then monitor the runs that typically last between 1 hour and 5 days.

vi) Job crashes are automatically cleaned and resubmitted.

vii) Staff will run the ‘summary’ script once the jobs have completed.

viii) Staff, using the generated summary script output, will perform housekeeping actions before publishing the data for export.

Strengths

• The failure rate of submitted jobs is relatively low at less than 10%. This is due to the system’s robust resubmission procedures.

• Production has been relatively constant despite many problems at regional centres. This reflects the system’s resilience to sites going down temporarily.

Current issues

• The system requires considerable manpower during normal operation. More efforts are needed in terms of automation.

• As the size of the datasets approaches 1 TB, the data transfer latency causes regional centres to experience delays to production as processed data is shipped elsewhere.

• Complex failure modes make it difficult to automate remedial action. Manual intervention is frequently required.
• The current job-decomposition scheme is an artificially imposed constraint based on one run per job. A possible improvement could be to take into account estimated running time or number of available CPUs. However, this further complicates both RefDB and Impala.

• The Objectivity database is the main cause of many of the problems experienced. A change in the data-storage model should alleviate problems.

• The use of tracking databases (BOSS and RefDB) has proven to be invaluable for statistical and optimisation purposes. However, there is considerable problem with respect to scalability and reliability when updating the databases in a hostile or overloaded environment.

GRID compatibility

The benefits of the GRID are difficult to ignore. CMS has since initiated many GRID prototypes for IMPALA, BOSS and RefDB. With the on-going CMS-DATATAG project, CMS has proven that although more work has to be done in the area of file replication, it has been able to submit and run its Monte Carlo jobs over the GRID successfully. There are also plans to integrate the EDG Scheduler with RefDB and to trial other GRID tools (e.g. LCFG, monitoring packages, etc.).

3.6 Request System (DØ)

Technology and architecture adopted

The DØ distributed Monte Carlo production mechanism, Request System, is one of the few grid-based production systems around. It consists of the following software components:

• mc_jobrun
  This is a suite of Python scripts that automate the execution of DØ’s Off-line software and interfaces with all the Monte Carlo generators. There are two basic components in mc_jobrun:

  i) The Configurator
      This component possesses the capability to run a requested DØ Off-line executable.

  ii) The Linker
      This component is responsible for instantiating and coordinating Configurators. It also handles the communication between Configurators and the user.

Figure 25 on the following page depicts the user issuing commands to attach and configure various Configurators via the Linker. Upon request (again through the Linker), each Configurator generates a shell script to run its corresponding executable. The Linker collects these individual shell scripts and compiles them into a master shell script before presenting it back to the user who then runs it.

• Sequential Access to data via Metadata (SAM)
  SAM provides for all of DØ’s data cataloging, storage and access needs which includes 60 institutions and over 500 physicists worldwide. It is the grid-enabling component in the Request System. Its distributed architecture is based on the CORBA framework. Systems on which SAM is deployed are termed SAM stations.

  The current SAM architecture consists of the following shared services:
i) The CORBA name service registers and acts like a switchboard for the entire distributed network.

ii) The central ORACLE database stores all metadata\textsuperscript{14} for the files in the system.

iii) The global resource manager performs request coordination and optimisation. It employs Fair Share scheduling and resource co-allocation to provide for the fair-sharing of resource and to maximise the overall work throughput, respectively.

iv) The log server collects and compiles log entry information from all stations into a central log file.

SAM stations primarily manage disk cache, job dispatch and resource allocation. All data and corresponding metadata (data descriptors) generated by the experiment are cataloged. Data required by a job is replicated by the SAM station to its local cache disks before the job is permitted to run.

Disk cache management in SAM is complex due to its distributed nature and 2-level cache structure (i.e. local and global caches). Locally, the SAM station works pro-actively to pre-fetch data for consumers (i.e. the jobs) whilst relying on sophisticated algorithms to perform data load balancing. Globally, not all SAM stations have direct access to Mass Storage Servers (MSS) and so data has to be stored via the data store of intermediate SAM stations that subsequently transfer the data to the MSS. This is effectively a grid replication service.

- Batch System
  
  Any\textsuperscript{15} existing batch system may be used to provide distributed processing capabilities. SAM communicates with the batch system to provide coordinated job process management and data delivery.

- Other essential packages include Linux\textsuperscript{16} operating system and the Python scripting language.

---

\textsuperscript{14}The format of the metadata stored has recently been changed from Metadata to the new MetadataSystem format.

\textsuperscript{15}SAM provides interfaces to LSF, PBS, FBS and Condor. Adding new interfaces is fairly straightforward.

\textsuperscript{16}Fermilab Redhat Linux 7.1
Production workflow

DØ’s Monte Carlo production workflow is well defined and well documented. To create a request, the following steps are needed:

i) Create a Python script\textsuperscript{17}. This script must describe all the variables to be used in the request.

ii) Submit the request (the Python script as the input) with the `sam_create_request` command which returns a request ID.

iii) Before the request is placed on to the job queue, it has to be approved by a local Monte Carlo representative.

iv) Requests may be monitored through a webpage\textsuperscript{18}.

GRID compatibility

DØ’s Request System / SAM is a GRID-like system. It is currently moving towards a much less centralised model in which stations have autonomy and are capable of operation without contacting the central database at Fermilab for long periods. Each station or site will have its own local information service that will track local project activity and communicate with the global information service. This decentralisation will remove the current single point of failure and greatly improve performance of the system. DØ is constantly monitoring EDG middleware for suitability to make the transition to standard GRID middleware whenever possible.

\textsuperscript{17}This is based on the `mc_jobrun` Python script suite. Users only need to be familiar with very basic Python syntax as a production template is available.

\textsuperscript{18}http://www-d0.fnal.gov/computing/mcprod/request_details/Request.html
3.7 SLICE (LHCb)

Technology and architecture adopted

SLICE is LHCb’s Monte Carlo production environment with built-in monitoring and control mechanism. The main components of the SLICE software are:

- Job preparation service
  Production job scripts (.sh or .bat), card files (.sicb.dat) and option files (.opts) are generated by servlets from four\(^{19}\) servletrunners\(^{20}\). These servlets are html-based allowing the user to submit jobs through a convenient web-interface (see Figure 27). There are currently four job-submission servlets (each with a different purpose and site-specific configuration) in use: maprmnc, brunelrun, bbinclrun and mcbrunel.

![MC Production Job Submission Form](image)

Figure 27: The SLICE job submission web interface.

- Control system
  The control system is based on the PVSS software which has a client-server architecture. The server script, mcserver, runs at every production site. Apart from allowing the local production manager to manipulate jobs, the PVSS client also issues warnings when errors occur at predetermined checkpoints.

- Book-keeping service
  The book-keeping database is updated by issuing the cdispos{e} script-based command.

- File transfer service
  The Java-based file transfer module, bbf{t}p{a}n{e}sfer, is used to automate transfer of data to and from Castor.

\(^{19}\)One at CERN, two at Nikhef/VU and the last one at DataGrid.

\(^{20}\)JSDK 2.0 for Linux.
Production workflow

![Diagram of the SLICE workflow](image)

Figure 28: The SLICE workflow.

The Monte Carlo production workflow can be summarised into the following steps:

i) The physicist issues a request for Monte Carlo production and he provides information that describes the particular production i.e. no. of events, channel, datatype, configuration, deadline for completion, etc. This request is submitted to the physics coordinator.

ii) The physics coordinator ratifies the production request which is then added to the set (a database) of outstanding requests. The request is sent to the production manager.

iii) The production manager executes the request via the web-based job preparation service. The required scripts are automatically generated.

iv) The production manager submits the scripts to the control system for execution and the output is stored in a pre-defined location ready to be retrieved by the physicist. The book-keeping database is updated automatically.

GRID compatibility

EDG middleware is currently been progressively incorporated into SLICE. Many of the SLICE components can be easily mapped onto corresponding GRID tools without fundamental architectural changes. In fact, the gridification of SLICE is barely noticeable to the physicist as the GRID-enabled job submission web interface\textsuperscript{21} (see Figure 29) is very similar to the pre-GRID version in Figure 27.

\textsuperscript{21}http://lhcb-comp.web.cern.ch/lhcb-comp/S1CB/pesf/html/testbedsub.htm
4 Conclusion

The experiments reviewed have shown significant interest in GRID technologies and recognise that it is a crucial step to take for current and future endeavours to be possible. Current non-GRID architectures, though less complex to maintain and operate, do not scale to levels required by experiments today. Moreover, it is simply not financially feasible for any individual site to provide all the computing power and storage needed for any particular experiment. The sharing of resources on a global scale is the only solution to this problem.

The common architecture that has been adopted by most of the experiments is a decoupled 2-tier architecture where there is a distinct front-end and back-end component.

All production systems involve many software components. This effectively translates into a multi-step production sequence that requires the expertise of a scarce few. Automation using scripts is a common way of alleviating the problem but there are limits as to what automation can offer. Current developments favour using a single front-end component that provides an interactive interface by means of a web interface or a thin client that runs locally. This component enables a non-expert user to configure a job, submit it, monitor its progress and collect the job output with little or no programming experience. All this is possible from a single environment and the user does not have to worry about how it is all done internally. Effectively, the interface hides the complexity that comes with dealing with the GRID.

These interactive interfaces also have APIs that allow experiment specific application developers to make use of the interface’s functionalities in specialised applications. Most of the interfaces reviewed have modular architectures to facilitate the extension of standard functionalities for customisation purposes.

The back-end mechanism is very much more complex and is not within the context of this review.

---

22This has almost become a requirement and definitely a highly desirable feature of any interface.
5 ACKNOWLEDGEMENTS

Although many experiments have their own version of GRID-like middleware, they (i.e. experiments in Europe) are gradually adopting EDG middleware because there is much duplication of effort in this area. Moreover, there are obvious benefits in terms of interoperability with adopting a standard middleware.

5 Acknowledgements

I would like to thank the following experts of the various experiments for taking time off to share their experiences with their respective experiments’ distributed production systems: Peter Faulkner(H1), Chris Hawkes(BaBar), Norman McCubbin(ATLAS), Armin Nairz(ATLAS), Steve O’Neale(ATLAS/Opal), Gilbert Poulard(ATLAS), David Smith(BaBar) and Tony Wildish(CMS). Many thanks also to Roger Jones and Peter Watkins for providing the much needed pointers to complete this review.
References

SAM Managed Cache and Processing for Clusters in a Worldwide Grid-Enabled System  

GENIUS User’s Guide  

BømGrid: Some UK developments  

How to Run DØ Monte carlo Generation  
http://www-d0.fnal.gov/computing/mcprod/software/howtorun.html.

How to Use the Monte Carlo Production System  
http://www-d0.fnal.gov/computing/mcprod/requestsystem/how_to_run.html.

mcp11 software release  

ALICE Production Environment  

funnel: Monte Carlo Production Using Hundreds of Computer Around the World: Towards Comfortable Large Scale Computing  

funnel: Towards Comfortable Event Processing  

U.S. ATLAS GRID Testbed Status  

mc_runjob webpage  
http://www-clued0.fnal.gov/mc_runjob/mainframe.html.
AliEn website  
http://alien.cern.ch/alien/.

Centipede User Guide  
http://www-h1.desy.de/~faulkner/h1_user_guide.txt.

skimData based Replica Catalogue  

GRAPPA - Experience in Using the Athena Notebook  

GANGA: Interfacing to Software Framework and the GRID  

Software Framework and GRID Interfaces: Technical Survey  

Survey of Technology Relevant to a user-GRID Interface for ATLAS and LHCb  

Adil’s notes from the BbB Grid Collaboration Meeting in London  

Simulation for LHCb and its Integrated Control Environment (SLICE)  

BdbServer++, Background, Progress and Future Plans  

SRB for BbB  

[23] Véronique Lefèbure.  
  CMS Note: The Spring 2002 DAQ TDR Production  
  http://cmsdoc.cern.ch/documents/02/note02_034.pdf.

  CMS GRID Integration  

[26] Pavel Nevski.  
  GRID Tools in ATLAS Production  

[27] Alois Putzer.  
  ATLAS Data Challenge 1 Phase 1 Project Report  

  LHCb Distributed Computing and the GRID  

[29] Alexandre Vaniachine.  
  Data Challenges in ATLAS Computing  

  Virtual Data and Production  

  BBo Monte Carlo Production SP4 Installation and User Guide  