Production of *BABAR* skimmed analysis datasets using the Grid

C. A. J. Brew[1], F. F. Wilson[1], G. Castelli[1], T. Adye[1], W. Roethel[1], E. Luppi[2], D. Andreotti[2], D. Smith[3], A. Khan[4], M. Barrett[4], R. Barlow[5], D. Bailey[5]

[1] Rutherford Appleton Laboratory, Chilton, Didcot, Oxon, OX11 0QX, UK
[2] INFN Sezione Di Ferrara, Ferrara, Italy
[3] SLAC, Stanford, CA 94309, USA
[4] Brunel University, Uxbridge, Middlesex UB8 3PH, UK
[5] University of Manchester, Manchester M13 9PL, UK

E-mail: `fwilson@slac.stanford.edu`

Abstract. The *BABAR* Collaboration, based at Stanford Linear Accelerator Center (SLAC), Stanford, US, has been performing physics reconstruction, simulation studies and data analysis for 8 years using a number of compute farms around the world. Recent developments in Grid technologies could provide a way to manage the distributed resources in a single coherent structure. We describe enhancements to the *BABAR* experiment's distributed skimmed dataset production system to make use of European Grid resources and present the results with regard to *BABAR*'s latest cycle of skimmed dataset production. We compare the benefits of a local and Grid-based systems, the ease with which the system is managed and the challenges of integrating the Grid with legacy software. We compare job success rates and manageability issues between Grid and non-Grid production.

1. Introduction

The *BABAR* High Energy Physics (HEP) detector [1] is based at the Stanford Linear Accelerator Center (SLAC), Stanford, US. The experiment investigates the subtle differences between matter and anti-matter by continuously colliding bunches of $\sim 10^{10}$ high-energy electrons and positrons 250 million times per second and searching for the creation and decay of rare B-meson and anti-B-meson particles. High speed electronics and online processing rejects unwanted events resulting in a final raw event rate to tape of approximately 100 Hz with each event requiring $\sim$ 30 kB. The raw data is sent to Padova (Italy), where the events are reconstructed. The reconstructed events are then separated ("skimmed") into approximately 250 data streams according to their physics properties using compute farms at Padova (Italy), Karlsruhe (Germany) and SLAC (US). These data streams are made available as datasets for analysis at Tier 1, 2 and 3 centers and are used by 522 physicists based at 77 institutes in 10 countries. The data streams result in increased storage requirements as each event is duplicated in different streams but each data stream can be analysed more quickly.

The disk space required to store the full dataset is $\sim$ 1.8 Petabytes (10^{15} bytes). The total processing power required to produce the current dataset and analyse it is $\sim$ 7000 kSpecInts in 2007 and $\sim$ 10000 kSpecInts in 2008. The major processing farms export and import data at a rate of ~ 1.5 Terabytes per day over the wide area network (WAN). Since starting in May 1999, the experiment has recorded 4.5 billion events.

2. Skimming

A major computing task is the allocation of each reconstructed event to one or more "skimmed" datasets based on the event characteristics. The skimmed datasets are of two types: "deep copy" datasets contain a complete copy of the original event while "pointer" skims contain only a pointer to the original event. "deep copy" skims can be exported to analysis farms which only require a subset of the data; the "pointer" skims can only be used at sites which hold the complete original dataset. Although skimming results in duplication of events in different streams and a consequent increase in storage space, the overall result is a reduction in resources as sites do not have to hold the complete dataset and researchers do not have to run their selection algorithms over every event.

The production of the skimmed datasets is controlled through two databases. A central book-keeping database [2] maintained at SLAC keeps track of the overall status of each collection of events and whether it has been successfully skimmed or not. A local production database is used to manage the production at the local site. The skim production follows the following cycle:

(i) At the start of a skim cycle, each production farm is allocated an initial set of events to process of approximately $10 - 20$ million events depending on the resources available to the farm. The availability of these events is indicated through the central database and an automatic process at the local site downloads the required events.

(ii) Using the local production database, the local production manager prepares a set of production jobs. The specification of the jobs includes the software releases to use, input datasets, conditions and calibrations to be read in, the number of events to allocate to each job, which skim datasets to output, and any local configurations. The local configuration includes such details as batch system requirements and the type of storage to be used. The details are specified through a configuration template file which is then duplicated for all the jobs. The use of a template allows local sites to change the configuration while still working within the overall production framework.

(iii) The jobs are submitted to the local batch system or the Grid. The submission is handled via the local production database which contains the status of the production. Jobs identified as failed can be resubmitted or abandoned. The submission is done via a set of scripts adapted for each site and batch system. They communicate with the production database through a common interface. This allows new sites or batch systems to be added to the system with no change to the overall framework.

(iv) Each skim job produces one file for each skim stream with up to approximately 250 output files per job. The number of events in each output file depends on the selection criteria for that skim stream. In general, skim streams that select more than 5% of the events are written as "pointer" streams. Most streams select $< 1\%$ of the input events and it is possible for some streams to select very few or no events per job.

(v) An independent merge step monitors the progress of the production through the production database. When enough events have been skimmed (typically $10 - 20$ million), a merge process is launched. This gathers up the files for each stream and concatenates them. This results in a considerable reduction in the number of files that must be stored in the mass storage system and in the number of records that must be maintained in the book-keeping database. Streams with many events are split into multiple files of ~ 2 Gbytes each. Each output file is checked for integrity and a checksum saved in the database to allow subsequent transfer errors to be detected.

(vi) The successfully merged files are exported back to SLAC. They are recorded in the central database, stored on the HPSS tape system and made available to analysts. At this point, all files produced at the local site during production are removed, freeing the local site for more production. When a sufficient number of events have been successfully exported back to SLAC, a new set of events is allocated in the central book-keeping database and the cycle continues.

The process described above is based on a non-Grid model of production. On each farm a complete *BABAR* software release needs to be installed locally or made available to the batch nodes via AFS [3]. A number of servers must be maintained locally: the location of the experimental and simulated data is provided by a MySQL [4] or Oracle database [5] database; the experimental and simulated events are stored and accessed using the ROOT [6] I/O protocol either directly through NFS or, for heavily used farms, with a load-balanced, fault-tolerant file server called Xrootd [7]. Detector conditions, alignment and calibration constants are distributed to the jobs in the same way. The system requires a great deal of local customisation with each farm maintained by a local manager. Load balancing between independent farms is coarse-grained, based on overall throughput.

The coming years will see a doubling in the experimental data rate, and a less manpower intensive system must be found to provide two times the resources. A possible approach is based on the tools and middleware provided by Grid systems such Globus [8] or the LHC Computing Grid Project, LCG [9]. The goal is a production system with a single production manager submitting jobs to a worldwide Grid of remote sites with automatic resource allocation, job monitoring, output retrieval and cataloging. Grid middleware, working over the Internet, provides the necessary infrastructure. Many Worker Nodes (WN) are managed by Compute Elements (CE) with jobs directed by Resource Brokers (RB), metadata management by the Replica Location Service (RLS) and high volume of data kept on Storage Elements (SE).

3. Skimming on the Grid

In 2006/7, a new framework (called the Task Manager 2 or TM2) for controlling the production of skimming was rewritten. The new design was influenced by lessons learned from previous years. Using object-orientated Perl, the goal was to produce a consistent framework that could be adapted for different sites through addition of plug-ins or local scripts. The requirements of a Grid-based submission system were integrated into the design from the start. The new *BABAR* Task Manager software was installed on the LCG User Interface (UI) at the RAL Tier 1 and properly configured.

The initial Grid system selected for production was LCG. LCG offers a number of possible advantages as the middleware was developed with HEP application use cases in mind. It provides an integrated data management system and a Resource Broker (RB) that can direct jobs to compatible resources without involvement of the submitter. Tests were also performed using the Open Science Grid (OSG) [11]; however only a few machines were available and we do not report on performance here.

The LCG model also adds a number of challenges. The *BABAR* software is not available at the remote site. Therefore the complete *BABAR* software was reduced to a core subset that was packaged as a tar archive of about 40 MB (130 MB uncompressed). This file included the executable, shared libraries and configuration scripts. The package was then distributed on each Grid Worker Node by using procedures based on the LCG middleware to install and uninstall new software and to publish new tags. The tags ensured that the executable would only run at sites with the necessary infrastructure and data. A site manager, mapped as a special user, was able to use these procedures by simply submitting specific jobs on the Grid choosing which sites must be involved.

Three sites (RAL Tier 1, RAL-PPD Tier 2 and Manchester Tier 2) were configured to provide the necessary resources for skim production including dCache or Xrootd services that could be accessed locally or over the Grid. However, access over the WAN was not efficient or reliable and we reverted to importing data to the local site. Access to the RAL Tier 1 was used to compare Grid and non-Grid submission as the batch queues could be accessed either via LCG or PBS [12] commands.

The standard production scripts were modified to submit remotely to the gateway; the jobs were passed onto the batch system and then retrieved at the end with GridFTP [13]. A JDL [14] script set up all data needed and specified the parameters for the simulation including type of events, run number, background triggers to use, number of events and detector conditions.

Jobs were then submitted through an UI to a Resource Broker specifically installed for *BABAR*, located in Ferrara (Italy). The RB was able to manage Italian and European resources directly involved in *BABAR*. The RB performed matchmaking between resources available on each site of the Grid, published by Computer Elements (CE), and jobs requirements (memory, specific software release and type of the batch queue). Jobs were sent to CEs satisfying the requirements, and distributed to WNs, where the execution started. Through the use of tags, jobs could be directed to the three sites where the events were available for skimming.

Local databases were setup to provide the conditions and calibration data while the background trigger files were either accessed via a local Xrootd server or preloaded onto the local Grid SE and copied to the Worker Node's local disk at runtime and accessed from there.

Grid production in the UK was managed from a central site (RAL). An allocation of events to be skimmed was requested from the book-keeping database at SLAC. The allocation would be divided in jobs of 5000 events which were created in unique sub-directories at RAL. The job control scripts, together with the executable, would then be submitted to the Grid. On completion, the output files, which included log files as well as the skimmed output ROOT I/O streams, would be packed into a tar file and transferred from WNs to a Storage Element (SE) located at RAL. From there, the output would be unpacked and transferred back to the originating subdirectory.

A checking and merging process ran continuously at the central RAL site. The checking process monitored the status of the files in the originating subdirectory. Since files in this directory would only change on submission and completion of a Grid job, the checking process did not have access to all the information that a standard batch system could supply. To reduce the number of files to be managed, the merge process would concatenate the output events into files containing $\sim 100,000$ events ($\sim$ 2 Gbytes) before being sent back to SLAC. Once at SLAC, the files could be re-exported to the collaborator's site for analysis.

The Grid imposed some constraints on our ability to monitor and merge the resulting output. In the standard operation the jobs would communicate their status to the book-keeping database at SLAC. As a fall back method, the log files would be intermittently parsed to monitor current status (events processed so far) or completion status such as finished or aborted. This relied on being able to access the log files directly or through the batch system. Both these methods were unavailable with the Grid as there was no guarantee that the WNs could access the book-keeping database through their firewalls and the log files were unavailable until returned to the submitting site. Although LCG does have a mechanism for querying the status of the job, the response time was very slow (up to a minute) and the response did not contain adequate information. This also prevented us doing the merging on the Grid as database access was imperative for this stage, reliability had to be 100% and the receiving computers at SLAC were not Grid-enabled for file transfers.

The solution for the merging was to run the merge process on a non-Grid system at a central site (RAL) with the necessary firewall ports to the database and access to the returned log and event files. For the monitoring, the workaround was to create the jobs on the central system before submitting them to the Grid and to copy the output back to the same place. Until the log files were copied back, the monitoring would know the job had been submitted but not how far it had progressed; when the job was copied back the monitoring would parse the log file and update the status. If the log file was not copied back within 48 hours the job was assumed to have failed and resubmitted. On the fourth failure, the job was abandoned.

The peak production rate was 30 million events per week with approximately 250 jobs running simultaneously on a single Grid site. The average success rate for the jobs to run first time was 50% with a low of 10% and a high of 97%. However the overall success rate including resubmissions only reached $\sim$ 90%. Problems tended to be due to catastrophic failures of the Grid infrastructure which meant that jobs that failed on first submission had a high likelihood of failing on resubmission. In particular, failures in the Resource Broker would result in all jobs failing. Since the system did not include a back pressure mechanism, the system would continue to submit jobs. The only solution was to stop the submission and identify the problem manually. This would result in a few days lost production.

Conversion of the existing job submission to submit to the Grid was relatively straightforward. Additional code was needed to ensure the right environment was maintained and that any additional software not at the site was transferred with the job. Although the same executable was run each time, the auxiliary files that determined the number and type of output streams could change depending on whether data or Monte Carlo events were to be skimmed.

The unreliability and slowness of the monitoring software meant that it was not possible to monitor the status of jobs once submitted. The next checkpoint only occurs if the job finished and returned information to the central site. If a job failed but did not return information, the job was assumed to be still running. Consequently, additional software and algorithms needed to be developed to recover from disappearing jobs. Even when the reliability of the job monitoring improved, querying the status of all the jobs was very expensive in terms of time, sometimes taking upwards of 30 minutes when large numbers of jobs were in the system. It was only when jobs status information started to be published via R-GMA [17] that near real time job monitoring was possible.

No software was available to monitor the status of Grid services such as RBs and catalogues. The ability to prevent job submission in cases where the intermediate services were not functioning would have saved much lost production. During the testing phase, the LCG software and middleware performed reliably. However, the introduction of new services over and above those that a standard batch system uses did introduce new points of failure. A brute force approach of resubmitting failed jobs mitigated some of these effects at the cost of increased use of resources.

In general, the ability to run at different sites did provide a more consistent production rate and load-balancing. However, the main focus was to develop production capabilities at a single site. No attempt was made to use Grid middleware to split the production across multiple sites.

4. Conclusions

The skimming of HEP events is now feasible using Grid/LCG architecture. Reliability is beginning to approach that of local self-contained compute farms. The Grid can be interfaced to legacy software and procedures. As most non-trivial HEP programs require access to centralized or distributed resources, our prototyping shows that the major architectural problems concern the installation of these servers and their efficient access over the WAN. Future work will concentrate on providing a common environment on all Grid resources and integrating the system with OSG.

References

[1] Aubert B *et al.* 2002, Nucl. Instr. and Methods A **479**, 1

[2] Smith D 2007, Data management in BaBar, CHEP Proceedings, Victoria BC, Canada.
[3] Howard J H 1998, An overview of the Andrew File System, Carnegie Mellon University, USENIX Conference Proceedings.
[4] MySQL AB, [online] Available: http://www.mysql.com
[5] Oracle Inc, [online] Available: http://www.oracle.com
[6] Brun R and Rademakers F 1997, ROOT – An object orientated sata analysis framework, Nucl. Instr. and Methods A **389**, 81
[7] Hanushevsky B 2004, The next generation root file system, CHEP Proceedings, Interlaken, Switzerland.
[8] The Globus Alliance, [online]. Available: http://www.globus.org
[9] CERN, LHC computing grid project, [online]. Available: http://lcg.web.cern.ch/LCG
[10] Ganglia, [online]. Available: http://ganglia.sourceforge.net
[11] Open Science Grid, [online]. Available: http://www.opensciencegrid.org/
[12] OpenPBS, [online]. Available: http://www.openpbs.org
[13] GridFTP, [online]. Available: http://www-fp.globus.org/datagrid/gridftp.html
[14] Job Description Language: DataGrid-01-TEN-0102-0_2, 2002, Datamat
[15] dCache, [online]. Available: http://www.dcache.org
[16] PNFS, [online]. Available: http;//www-pnfs.desy.de
[17] RGMA, [online]. Available: http://www.r-gma.org