The *BABAR* Analysis Task Manager

W. Roethel, *Univ. of California at Irvine, Irvine, CA, USA*
D. A. Smith, A. Ceseracciu, *SLAC, Menlo Park, CA, USA*
T. Adye, *RAL, Chilton, Didcot, Oxon, UK*
D. Bukin, *Budker Institute of Nucl. Physics, Novosibirsk, Russia*
G. Dubois-Felsman, *Caltech, Pasadena, CA, USA*
A. Forti, *Univ. of Manchester, Manchester, UK*
D. Hutchcroft, *Univ. of Liverpool, Liverpool, UK*
P. S. Jackson, *Univ. of London, Royal Holloway, Egham, Surrey, UK*
D. Kovalskyi, *Univ. of Maryland, College Park, MD, USA*
For the *BABAR* Computing Group

Abstract

*BABAR*s Computing Model 2 introduced a new event store and new framework for the bookkeeping of *BABAR* data. The *BABAR* Analysis Task Manager, as part of the new bookkeeping framework, acts as an interface between the information stored in the *BABAR* bookkeeping, the data served by the event store, and the offline framework. It provides the basis for large scale processing of *BABAR* data, which is necessary for data production or data analysis.

INTRODUCTION

In the new Kanga event store [1] *BABAR* data is organized into collections of events. These event collections provide an interface to an underlying set of ROOT files which contain the data itself. Since the collections are based on ROOT files, data access is not tied to any particular server technology. However simple network file systems can not maintain the requirement of providing data concurrently to several thousand jobs. *BABAR* has therefore developed xrootd [2], a high-performance, rootd compatible file server which satisfies all requirements for stability, scalability and performance. The new *BABAR* bookkeeping [3] keeps track of all collections stored in the event store. Access to these collections is made over predefined datasets, which are sets of collections to be used for data analysis or data production, in particular the general production of skimmed data, which most *BABAR* analyses depend upon.

Life Cycle of BABAR Data

Any *BABAR* analysis or skim production typically follows the following steps:

- Selection of one or more suitable datasets providing the list of collections used as input for processing.
- Define jobs to be run on the computing farms to process the input and write output data to temporary location.
- In a post processing step, merge the output of several jobs into larger data files.
- Import the merged collections into the event store and insert information on these collections into the bookkeeping database.

A summary of the life cycle of *BABAR* data is given in Figure 1.

The Task Manager provides an automated interface that deals with managing and bookkeeping of analysis and production jobs including the creation, submission, and validation of jobs and the subsequent post-processing. It consists of two major, independent components, the bookkeeping framework that provides the user interface and is responsible for the bookkeeping of jobs and job output, and the job processing framework, which deals with running and validating the particular jobs in a scalable environment on the computing farms.

THE TASK MANAGER BOOKKEEPING FRAMEWORK

A task in the context of the Task Manager defines the overlaying unit in which the single jobs are managed. Every task is defined over a configuration which is common for all jobs within a task and stores information on the processing itself ('How' the data should be processed, e.g. the application to be used for processing, the global output location of the data, etc.), and one of more datasets ('What' should be processed). Both, the configuration and the datasets to be used in a task, are updateable. This leaves it up to the user to define the actual boundaries of a task. As an example a task could be defined to process all data for a single analysis or for several analyses. The flexibility in the configuration also allows users to continue processing with a new version of their analysis software without requiring them to re-process data already processed with the previous release. The Task Manager creates the jobs to be run on the computing farms based on the list of input collections provided by the defined datasets and requirements specified by the configuration, e.g. the maximum number

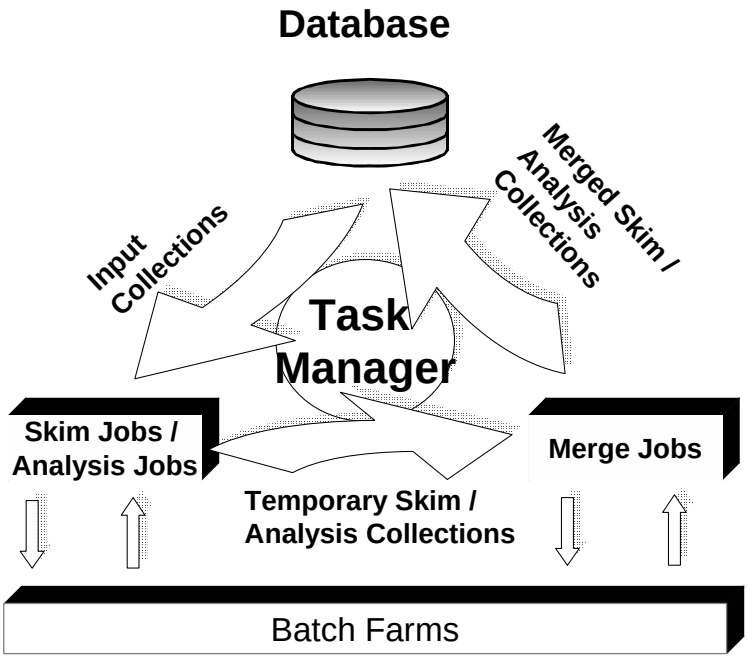

Figure 1: The Life Cycle of *BABAR* data collections. Collections are used as input for data processing which creates new collections that are put back into the *BABAR* event store and themselves can be used as input for further processing.

of events to be processed per job. At any time all jobs are in one of many different job states. Initially all newly created jobs are in the 'prepared' status, i.e. these jobs are ready to be run on the computing farms. Other common job states are 'submitted', a job has been submitted to the local job scheduler to be run on the computing farms, 'ok', a job has completed successfully and has been validated, and 'failed', which indicates that the processing of this job has failed. Other job states can be defined by users if found to be necessary. The scheduling of the jobs on the computing farms is handles by the local job schedulers. The Task Manager provides an interface to these (currently LSF and PBS are supported) where job monitoring features are provided by storing the ids of the jobs as used by the job scheduler. The Task Manager does not directly execute the application on the computing farms, but rather hands the execution of the processing over to a job wrapper. The job wrapper is responsible for setting up the necessary environment for processing and validating the success of the processing. The job wrapper will be discussed in detail in the following section. When jobs have finished running on the computing farms, their job status is updated according to information provided by the job wrapper. Failed jobs can be resubmitted if necessary and a basic versioning system keeps track of the various different processing attempts.

Post Processing

In the current version of the Task Manager the merging of the different, rather small, output collections of different analysis or skim production jobs is managed by a separate task. The configuration is similar to the underlying analysis or production task, the major difference being that the input to the merge task is not defined over a dataset but over the output of another task. This is somewhat cumbersome and in the updated version of the Task Manager, which is currently under development, the post-processing will be an integrated part of the main analysis or production task itself. The Task Manager includes tools to import the output of the post-processing into the event store and insert the necessary information on the data into the *BABAR* bookkeeping.

THE TASK MANAGER JOB PROCESSING FRAMEWORK

A crucial requirement for running analysis and production jobs is scalability. Data production may need to run more than 1000 jobs concurrently. This is achieved by defining a run environment that uses local resources whenever possible. In particular, jobs are not allowed to contact relational databases from the computing farms, and all data

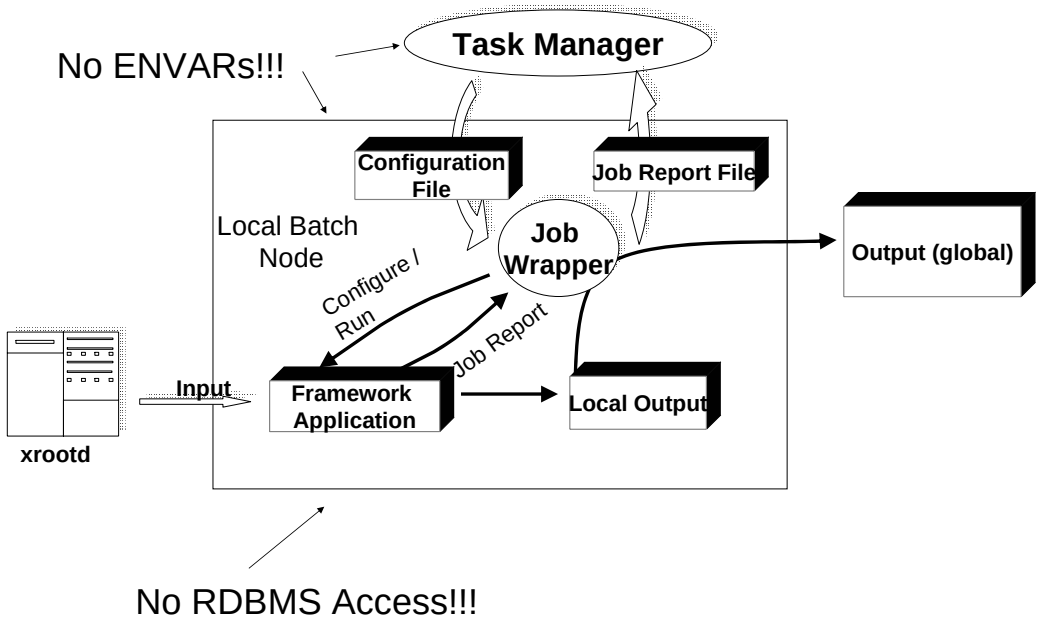

Figure 2: The configuration of the Task Manager Processing Framework. Jobs are required to run in a locally confined processing environment to allow for scalability.

output is written to local disk on the processing nodes and only after completion of the processing and validation of the output is the data copied to a global location. Input data is served over xrootd and poses no problem in terms of scalability. Setting up this local environment is one of the responsibilities of the job wrapper. Information on the particular configuration of an individual job, e.g. the name of the application and the final, global output location of the data, etc., is passed on to the job wrapper over a job configuration file. After managing the execution of the application, the job wrapper verifies the success of the processing. This is done on different levels beginning with the exit code returned by the application itself up to the validation of the integrity of the output collections. The main source of information for validation is the job report file, which is created by every *BABAR* offline application. It contains all important information on the processing in a standardized format, including information on data input and output, and a summary of errors and warnings. Provided the success of the processing is verified, the job wrapper copies the output to the assigned global location, collects all information on output and processing, including error codes and warning messages, which should be passed on to the Task Manager bookkeeping framework, and creates a job report file itself. This job report file is used subsequently by the bookkeeping framework of the Task Manager to update the job status. Figure 2 summarizes the processing framework as it is used by the Task Manager.

DISTRIBUTED PRODUCTION

The standard bookkeeping tools for dataset management and data distribution allow the Task Manager to run in a distributed environment. Every local production site requires its own dataset which contains a list of collections to be processed at the site. The dataset is used to drive the necessary data imports to the site and define the input to the production tasks. The output collections of the production are transferred back to the main event store and inserted into the bookkeeping using the standard Task Manager tools. The skim production currently is using this system of distributed production by sharing production efforts between SLAC (US), GridKa (Karlsruhe, Germany) and Padova (Italy). A summary of the distributed production is shown in Figure 3.

SUMMARY AND OUTLOOK

First versions of the Task Manager were used in November 2003. Since then in particular the processing framework has undergone many changes. Throughout 2004 it has been successfully used in the production of skimmed

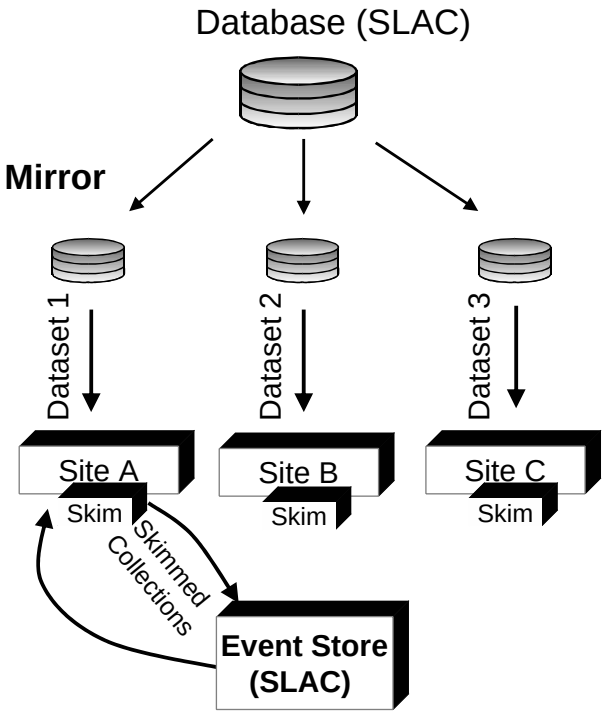

Figure 3: Distributed production in the context of the Task Manager. Local production sites use designated datasets and local database mirrors to import the required input collection and configure the production tasks. Output of the production is transferred back to the main event store at SLAC.

data, which provided the basis for many results presented at ICHEP 2004 [4]. To date over 100,000 individual skim production jobs have been managed by the Task Manager, with up to 1200 jobs running concurrently. Currently preparations are underway for the next round of skim production beginning in Nov. 2004. Improvements will include easier job configuration, full bookkeeping support for processing single collections with several jobs, and integrated support for post-processing. Looking beyond that the possibility of running the Task Manager in the GRID is an interesting option. The locally confined job wrappers make the Task Manager an ideal candidate for running in a distributed environment. Local *BABAR* sites would be selected based on availability of required software releases and input data collections. The validation of the processing and the transfer of the output data to assigned locations would be provided by the job wrapper itself. Distributed production would be run by a single task at a central location and would require only one operator to manage the effort. This would resolve the need of having one operator for every local site, as is currently the case.

REFERENCES

[1] M. Steinke, P. Elmer, *et al.*, "How to Build an Event Store - The New Kanga Event Store for *BABAR*", Proceedings of CHEP04, 2004.

[2] A. Hanushevsky, A. Dorigo, F. Furano, "The Next Generation Root File Server", Proceedings of CHEP04, 2004.

[3] D. A. Smith, *et al.*, "*BABAR* Bookkeeping - A Distributed Meta-Data Catalog of the *BABAR* Event Store", Proceedings of CHEP04, 2004.

[4] "32nd International Conference on High Energy Physics", http://ichep04.ihep.ac.cn .