

BaBar Analysis Grid application - Use Case and Requirements Document

Draft: 4. 5/Oct/01

BaBar Analysis

This document describes a BaBar Grid based analysis application which we believe should be achievable on the timescale of 1 year.

The BaBar analysis structure involves running a program on a set of data files. These programs select events of the required type, and output specified quantities to small 'ntuple' files.

The program fits into the general Babar framework ('Beta') but the code is written by the user for their particular purpose. Some element of flexibility may also be given through specifications in control files ('tcl files').

The data may be stored as flat files using root or using the Objectivity database.

The complete dataset is very large, comprising thousands of files and Terabytes of data. The program is accordingly submitted many times, i.e. as many jobs, each running over a subset of the data. Processing these jobs in parallel using many CPUs gives a valuable saving of time.

The ntuples produced by the different jobs are merged to give final histograms, numbers etc. using the packages PAW or ROOT. (This stage of the job could also be done in parallel.)

The Application

We have already constructed a very simple prototype which uses some aspects of the Globus software. Details of how this operates are given at the end of the document. The demonstrator application we propose is an evolution of this which will require the incorporation of several more aspects of Grid middleware technology.

The user will be presented with a GUI (or, if they prefer and probably in the early stages of the project, type a command) which will allow them to locate data files in root format ('kanga files') on the BaBar Grid against some BaBar meta data query. This is presumed to be done in a Grid like way, using Grid replica catalogue tools where they exist.

For this application it is assumed that the files have been made to exist at different sites by some other (possibly non Grid) means.

The GUI will then allow the user to submit a set of jobs to remote sites capable of processing the data files. In this initial instance we define that to be the site at which the data resides (so no WAN access will be required for data)

Wherever possible Grid like file transfer will be used for data and any ancillary information. Only where it proves to be difficult or impossible will we use existing native AFS access.

At the end of the analysis all the results from all of the jobs will be concatenated and returned to the output location specified by the user. Alternatively, the user may wish to control this themselves.

Use Cases

These take very specific and complete walks through a possible use of the application which has been described. In a general problem one is supposed to look for 40-80 use cases - but in HEP we seem lucky to find 3 or 4. The experiments should think hard though. It is use cases that bring out a lot of the specific functional requirements which are not apparent from a high level view.

Use Case 1: Standard Physicist Analysis

U1.1 The user will authorise upon their local site. At this point everything which is necessary to have available the credentials to run jobs at any BaBar UK Grid site must be done. In this first case BaBar UK will not differentiate between users in any way, provided they are on a list of 'Babar users'. The local site can be anywhere, provided the user has a certificate recognised by the remote site(s) they are going to use.

U1.2 The user will prepare a piece of C++ analysis code at their local site, where they compile, link and test it. This is done in the 'normal' Babar style, i.e. from within a workdir obtained by CVS checkout (no grid involvement occurs at this stage; the workdir must be part of an afs directory). The purpose of this is to ensure the code works as they wish and to produce an image which is to be run at a remote site or sites.

U1.3 The user then wishes to invoke a Grid front end GUI. This does not require a further authorisation step. This will be used to prepare and submit the job to an arbitrary BaBar Grid site as discussed below.

U1.4 The GUI first provides a unique identifier to the user, by which they will be able to refer to the analysis run at any later stage.

U1.5 The user will specify the meta data to identify the required input files and configure BABARANALYSIS. Fields specified may include:

- Data selection tags
- Date of data taking

- Reconstruction program version
- On peak/off peak
- Status of data (i.e. whether certified 'good')
- Drift chamber high voltage
- Real or Monte Carlo data
- Explicit run numbers, or ranges thereof
- Max number of events for a single job

U1.6 The GUI must then find the physical file location references corresponding to all of the files which satisfy this meta data specification. The physical files may be anywhere on the UK BaBar Grid sites. In this first phase BABARANALYSIS can access data at the granularity of a FILE (i.e the access is not object based). If there are more than one version available the GUI should decide which to use, on the basis of site loading, convenience, the accessibility of the data there (e.g. on disk or on tape) and information about the user.

U1.7 The GUI will generate a set of tcl files accordingly, each containing the event data filenames.

U1.8 The system must prepare a distinct job to run each tcl file. Therefore for each input file it must prepare a configuration tcl file which includes:

- Input file(s)
- Unique temporary output location(s) for the intermediate results
- Some specified environment variables (eg. \$BetaMCMicro, but not \$BFSITE)
- Other (standard) configuration information

U1.9 The user tells the GUI to submit the jobs. The user now logs out of the GUI.

U1.10 The system must, for each job, locate the appropriate BaBar Grid node upon which to run the job. In this first phase the algorithm required to do this is trivial: the job is to be run at the same site as the physical file location . It may be assumed that the correct environment exists at the site (i.e platform, release version, DLLs..)

U1.11 The system must then run the job at the remote site. This involves transfer of

- The configuration file
- The image to be run

U1.12 When the job runs it can be assumed that all date input files are accessible using the locally mounted BaBar file system \$BFROOT/.....

U1.13 When the job runs it will need to access further remote tcl files which have been specified in the configuration information. These are available via AFS.

U1.14 When each job terminates all of the output files must be copied to the destinations set by the user when using the GUI.

U1.15 When all jobs have terminated the system must, if required, run a local concatenation job which knows how to combine all of the output files.

U1.16 Finally the process terminates, and the final output files are copied to the location originally specified by the user.

U1.17 At some time later the user logs back on to the GUI and looks at the job status until the job is complete.

Use Case 2: User Status monitoring

This is an extension of Use Case 1 to capture the requirements of logging and monitoring.

It is assumed for this that a user has submitted an analysis as per Use Case-1

U2.1. The user logs back into the GUI.

U2.2. The user identifies the analysis which was submitted using the unique identifier system.

U2.3. The user interrogates a status display which shows the status of each of the jobs submitted as part of this analysis.

U2.4. The display should show whether jobs have failed, and if so gives an indication of why.

U2.5. The user may chose to disregard any failed jobs (i.e. remove then from the analysis using the GUI) or resubmit them.

U2.6. The user should get information as to how jobs are progressing, so they can see if they get stuck, and know when they are near completion.

U2.7. The user will be informed by the GUI when the entire analysis is complete, and all files have been concatenated and put in the specified output location.

U2.8 Log files from the jobs (stdout and stderr) will be stored on the remote node in a scratch area. The user is able to copy them back to their home computer if desired.

Use Case 3: Data import

U3.1 The command should select the data to import as in U1.5

U3.2 The command should know how much space is available locally and put the data in the file systems that have more space.

U3.3 The program should know what is the status of the data at the source on disk, on tape, deleted.

U3.4 The program should be able to use different protocols

U3.5 The program should be able to restart from where it left if there is some failure.

U3.6 If there are multiple sources the command should decide what is the best one depending on network traffic conditions.

U3.7 The program should distinguish what type of data (kanga files, kanga pointer collections objectivity)

U3.8 the protocol used should depend also on the type of data

Use case: repeat analysis

U4.1 The user examines the results of an analysis, as in Use case 1,

U4.2 They find an anomaly, and as a result discover a bug in their code

U4.3 They correct the bug, re-make the binary, and test it

U4.4 The user uses the identifier (see U1.4) to re-run the same analysis

Use case: site problems

U5.1 The user submits a job (as in use case 1). However jobs that run at a particular site all die for some subtle technical reason peculiar to that site.

U5.2 The system is not smart enough to realise this and continues to submit jobs to that site. (It may even favour it, as the load is light!)

U5.3 The user is smart enough to realise this, and has the information.

U5.4 The user does an instant-resubmit on the failed jobs, specifying that they must not run at this site.

Further Use cases

It would be instructive to write down use cases for

MC production

Analysis using Objectivity

Analysis of (large) ntuples to get final numbers.

Requirements

The following requirements are written down either as a combination of absolute requirements which are known outside of use cases, plus those additional ones which derive from the use cases themselves.

R.1: An authentication and authorisation system must be available at every BaBar Grid site which allows an inbound request from a valid BaBar user to be able to access all local resources assigned to BaBar.

[We imagine this to be provided by adoption of the A.McNab account leasing system]

R.2: Data files stored on the UK BaBar grid are in Kanga format. Therefore the Grid middleware must be able to transport this format.

R.3: Data files are stored on the local storage systems at RAL and the Babar universities (Edinburgh, Liverpool, Manchester, Birmingham, Bristol, RHUL, Brunel, QMW and IC). Therefore the Grid middleware must be able to catalogue and access all of these systems.

R.4: A method of identifying the environment(s) which exists at remote sites must be provided, such that the environment under which the locally compiled image can be compared for match.

R.5: The GUI must provide and manage a method of uniquely identifying an analysis job for the purposes of logging and status checking.

R.6 A BaBar specific “meta data” and “data file” cataloguing system must be provided, which allows the user to query (via the GUI) against the attributes of U1.5. Some of these are general, some are experiment specific. From the query it must retrieve a list of physical file locations for matching data. The catalogue must be able to index all files spread in an arbitrary way across the BaBar Grid. This means it must know which all the Babar UK Grid sites are. It must be able to specify the ‘best’ site if there are alternatives. (The user must be able to over-ride this) It should be compatible with the existing skimData scheme, probably using an ldap or spitfire front end.

[Note we imagine this could probably use the Globus LDAP replica catalogue scheme, and the LFN, PFN, TFN file naming schemes]

R.7: The replica cataloguing system must be capable of referencing the root data file type and disk (everywhere) and tape (at RAL) storage mechanisms.

R.8: The system must provide a way of specifying the following information which

must be transmitted to the remote site in order to specify the job to be run

Input files(s)

Output file

Location of Image to run and something to specify the environment it must run under

Location of job options file (text file containing arbitrary strings meaningful only to analysis application)

[Note: we assume this will have to be transmitted in a JDL format and so this list helps ensure that this can be done, or if JDL is inadequate then an alternative is found]

R.9: It must be possible for the executing job to access text files from remote Grid nodes. These (.tcl) files may either be known at configuration time (i.e. set during the initial user interaction) or generated dynamically by the job (text files may specify other text files.)

R.10: A logging system must be provided which allows all components to register the status of the job. In particular the site, the queue position, run status, and information to indicate any failure mechanism. This information should be available to the user whenever they log on to the system.

[Note: I know nothing of logging and presentation ideas. Need a work package expert to advise on this]

R.11: The system must provide a way of collecting the output from all jobs when all have terminated (allowing also for failures) and running a concatenation program. This can be executed anywhere.

Additional information

Details of operation of existing prototype:

0. The user obtains a BaBar grid authentication certificate, and accounts on two nodes (called 'local' and 'remote' below). This, unlike steps below, is done only once.

1. The user prepares a piece of C++ analysis code on the local node. This code is compiled and linked into the standard code at the local site. It is also tested there.

2. The user chooses a site at which they wish to run the job. This operation is done by hand and is outside the scope of this use case. The site is identified by a DNS name where the remote job submission interface resides

3. The user queries (by hand) the replica catalogue at the remote site, giving a description of desired properties for micro data as query and creating a set of files as response. These files ('tcl files') are in the current directory on the local site (the 'work directory'). Each of these tcl files contains a number of filenames of event data. These

filenames are universally valid (they are of the form \$BFROOT/data/... where BFROOT is a site dependent environment variable set in the profile script) and typically use nfs. The RC at the remote site is maintained locally by the site admin, and only knows of files at that site.

4. The user uses a script to prepares a unique set of jobs for the chosen remote site, one for each of the tcl files returned.

5. The script submits all these jobs to the remote site using globus job submission services.

6. At the remote site each job is recieved by the resource allocation manager. The globus certificate is authenticated and mapped to a unique UID corresponding to the user. The job is scheduled in the normal way.

7. The job accesses input control tcl files, of which the list of event filenames is one. These can be nested - file foo.tcl may direct the program to read file bar.tcl. This requires the running job to see the correct directory tree structure. For this reason the job script (on the remote site) changes its working directory to the workdir directory (on the local site) using afs.

8. During operation the job accesses the event data files specified in the job. These are on the same node, or connected to it by fast LAN.

9. Output is written to an ntuple file (hbook or root) also on the local site in the workdir directory

10. Upon termination there is no notification back to the user. Merely the output file is kept at the specified PFN.

11. The user periodically uses globus job status services to determine when all jobs have finished.

12. Output from all the files is analysed using the PAW chain command (or root equivalent).