BABAR BOOKKEEPING – A DISTRIBUTED META-DATA CATALOG OF THE BABAR EVENT STORE

D. A. Smith, A. Ceseracciu, *SLAC, Menlo Park CA, USA*
T. Adye, *RAL, Chilton, Didcot, Oxon, UK*
D. Bukin, *Budker Inst. 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. Jackson, *Univ. of London, Royal Holloway, Egham, Surrey, UK*
D. Kovalskyi, *Univ. of Maryland, College Park, MD, USA*
W. Roethel, *Univ. of California at Irvine, Irvine, CA, USA*

Abstract

A number of changes were requested in BaBar's computing model 2 plans over the past year, and one of those changes was for a new meta-data catalog to go along with the new event store. The catalog needed to be flexible to handle all BaBar data, distributed to all BaBar sites, and fast access in user interaction. In the development of this catalog ideas about what is experimental data were discussed, and some new concepts were introduced as part of the system. Once the catalog was implemented new method were developed to support the requirements. The requirements, concepts and developed methods are discussed in the paper, along with comments on how the system has worked in production.

For any questions or for further information please contact the principle author (Douglas Smith, e-mail -- douglas@slac.stanford.edu).

BABAR COMPUTING MODEL 2

In the beginning of 2003 BaBar had decided to implement a new computing model, which required a number of changes to the computing effort in the experiment. The biggest change was redesigning the event store, to be based on root IO and root files and not on Objectivtiy databases. Prompted by this change in the event store format was a stated desire for a new meta-data catalog. The meta-data catalog requirements were: to be flexible so to provide lists of data for any possible analysis within the BaBar experiment; to be performant, so it would not get slower as the amount of data or analyses increased in the experiment; to be distributed to the rest of the BaBar collaboration sites, so any user could access a local copy of the system at their site. These requirements from the new computing model have been incorporated in the new BaBar Bookkeeping, a set of tools and libraries which interact with a relational database to provide lists of the required BaBar data. The BaBar bookkeeping supports both Oracle and MySQL as underlying SQL engines since they were already widely used in the collaboration for other purposes. This allowed a smooth introduction of the new bookkeeping and no particular extra effort for its maintainence. More information on the Computing Model 2 changes can be seen in the plenary talk by Peter Elmer [1].

CORE CONCEPTS

Collections

In the computing model the event store is made up of "collections". These are simply collections of events. The collections are independent from event format and production system. Each collection name is unique, and this provides the key for the analysis programs to be able to access the data. The unique name can consist of any string of characters, as long as it is unique in the event store. This is the heart of the bookkeeping database: a list of all the unique collections in the BaBar event store.

Each collection has a list of associated attributes to allow an appropriate means of selection of collections for use. Examples of these attributes are: simulated or real data, run cycle, data quality information (good or bad), all together there are about twenty of these attributes. These attributes can be either collection attributes, for example the number of events contained in the collection or they can be related to the runs the collection has been created from, for example run cycle. This is reflected in the way the attributes are stored in the database according either to a *1 to 1* relation with the collection names in the same table, or if there are shared attributes which need more information, these are stored as a *1 to n* relation in a separate table. Some of the information has been duplicated in the collection table in a *1 to 1* relation to produce a better performance of the queries. The larger disk space usage was considered acceptable compared to the gain in performance.

Example 1 : Collection names in the BaBar event store:

from event processing : /store/PR/R14/AllEvents/0004/96/14.4.4e/AllEvents_00049689_14.4.4eV01
from simulation production : /store/SP/R14/001237/200407/14.4.3a/SP_001237_016169
from skimming production : /store/PRskims/R14/14.4.3d/AllEvents/13/AllEvents_1317

Some examples of actual collection names in the event store are listed in Example 1. The collection names chosen in BaBar are based on a file path name structure just for convenience, it is important to note these are collection names, not file names.

Each event in the BaBar event store is made up of different components, and these components can be saved in different root files. Each collection is stored in one or more root files and saved in a *1 to n* relation in the database. Collections in the event store are organized to optimize data archiving and data distribution. To achieve this goal, similar events are merged together into single collections to create file sizes just under 2 GB, when possible. For data distribution purposes the information about the files associated with a collection are stored the bookkeeping. This constitutes the file catalog part of the bookkeeping and refers only to Logical Files Names (LFN), there is no information about the Physical File Names (PFN).

LFNs are "global" file information of the event store, and they are independent from any particular BaBar computing site or server. The details of the file access at any computing site are not kept in the bookkeeping, they are resolved by the application data access system.[1,2]

In the bookkeeping the meta-data about the events and their internal organisation is decoupled from file location, storage, and access protocol. All an application needs from the bookkeeping is a list of collection names, the files associated with those collections are located by the data access system, which is configured by the local computing center. This reduces the amount of meta-data that needs to be stored, and allows computing sites freedom to serve files in the best way possible for that site, and to change the location of their files without the need of updating the meta-data system. For more detailed information about the event store see [2].

Runs, and relation to collections

The data measured by the detector are organised into runs. Each run corresponds to a set of events measured under the same detector conditions in a certain interval of time. Although for years the run was considered a unit of data, in the new computing model it has been replaced by the concept of collection. Runs need to be processed, and they can be processed multiple times in the experiment. Further skimming can be performed over data from many runs. The event store needs to have the freedom of merging data as required to simplify storage, producing collections with events from any number of runs. This makes the relation between runs from the detector and collections in the event store an *n to m* relation.

Runs are nevertheless important for analysis because they are set of unique events. Since only one copy or version of data from any run is allowed to appear in any given analysis, this makes a run the basic "non-overlapping unit" of data management, which needs to be recorded in the bookkeeping. In any experiment there will be this non-overlapping unit, and it could depend on the experiment as to which quantity this is, either event, run, or some unit of time. But because of processing this original unit is not a unique value in the event store. The bookkeeping system has been designed to record the non-overlapping unit and the association with the collections in the event store. As an example, the skimmed collection -- /store/PRskims/R14/14.4.3/AllEvents/AllEvents_1317, contains processed and skimmed events from 23 different runs. One of these runs is numbered 49670, which is also in 126 other collections. These multiple collections come from processing and the different streams from skimming. The *n to m* relation in this example is about average in the event store, although some skim collections contain events from hundreds to thousands of runs.

Datasets and user access

End users don't want all detailed and confusing information in analysis, they want a well defined list of data (collections). This well defined list is provided in the bookkeeping system as a dataset, and in the basic form a dataset is just a list of collections. Datasets are produced as lists of collections with similar attributes: i.e. real or simulated data, run cycle, on- or off-peak, simulation decay mode and so on. Each dataset in the system has a unique name, and this provides simple and fast access to the lists of data. Users for analysis only need to know the dataset name they want, and most analyses require between two to six datasets.

Example 2: A very simple example of a dataset, which contains a list of only 2 collections. Datasets can contain any number of collections, in practice they contain lists from one to tens of thousands of collections.

```
prompt> BbkUser --dataset SP-uds-AllEventsSkim-Run4-R14 collection
COLLECTION
/store/SPskims/R14/14.4.3d/AllEvents/00/000998/200310/AllEvents_000998_1539
/store/SPskims/R14/14.4.3d/AllEvents/00/000998/200309/AllEvents_000998_1540
2 rows returned
```

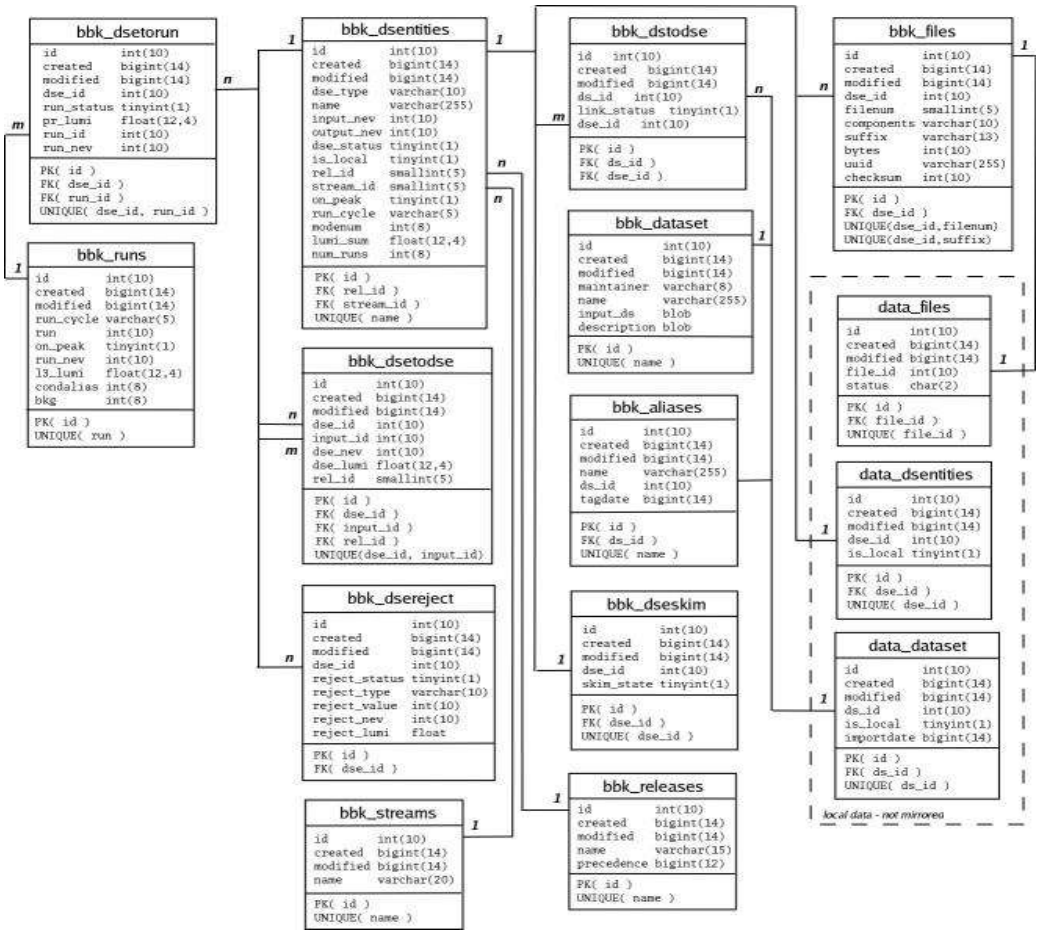

Figure 1 : The database schema for the BaBar bookkeeping database. The database is more than just three tables for runs, collections, and datasets, but still as simple as possible.

Datasets are not static during production, they have to evolve. The bookkeeping has to support the ability of production to keep on publishing new collections as soon as they are accessible. As data continue to be measured by the detector, new collections are published into the bookkeeping by processing and skimming systems. These new collections need to be continuously added to the datasets, the list that results from querying the bookkeeping for a certain dataset can change frequently.

Bookkeeping has to support also the removal of collections from a dataset and guarantee at the same time that a user can recover the dataset status exactly as it was prior the collection removal. The need of removing collections is associated with quality checks and reprocessing or re-skimming of collections. These changes should be freely made by responsible groups at any given time. Once a collection is recognised as bad, it should then quickly come out of a production dataset.

But analysis groups need stable lists. They want to know that the list of collections they used in the analysis will still be recorded in the bookkeeping for the life of the experiment. The bookkeeping system was designed to support both requirements of rapid evolution and stable lists. All changes to each dataset are recorded in the database, and when the list of collections is selected, the complete list of changes is applied to the selection. To provide stability, a dataset can be selected up to a chosen 'cut off' time, so only the changes up to that time are applied further changes to the dataset are not.

At times stable lists of collections need to be announced to the collaboration, in a way that all analysis know they are using a similar set of data before publication. These stable announced lists are provided by tagged datasets. The tag of a dataset is just a name aliased to a cut off time, so this tag can be used to always select a stable list from a dataset at any point in the future of the experiment. This model of tracking all changes and providing named stable tags is similar to CVS.

The database schema for the bookkeeping which stores the meta-data are displayed in figure 1.

DEVELOPED TECHNOLOGIES

SQL selection API

The database schema as defined can be setup on any relational database system. There was a stated requirement that the database had to be supported on either Oracle or MySQL, but these systems use slightly

Example 3 : an example of dataset tags. In this case the dataset AllEventsSkim-Run4-OnPeak-R14 was increasing in size as the BaBar run 4 cycle progressed. Analysis needed to have comparable sets of data to present before the run cycle was complete, the dataset was periodically tagged through the run. These tags were named: “GreenCircle”, “BlueSquare”, and “BlackDiamond”, and show how the number of collections in the dataset increased as the run progressed. The current dataset (as of the preparation of the presentation) contains 80 collections.

```
AllEventsSkim-Run4-OnPeak-R14-GreenCircle     -- 44 collections
AllEventsSkim-Run4-OnPeak-R14-BlueSquare      -- 66 collections
AllEventsSkim-Run4-OnPeak-R14-BlackDiamond    -- 76 collections
AllEventsSkim-Run4-OnPeak-R14                 -- 80 collections
```

different flavors of the SQL language. Another difficulty was the requirement that users should be able to query the database with whatever type of selection without really wanting to know about neither SQL nor the underlying database schema. This meant for the developers to maintain an unsupportable list of specific SQL statements for each database interaction.

To solve this problem, a database selection API was developed, which would produce the SQL statements separately for each database system, based on simplified selections. Each column in the database tables was given an alias, and these aliases can be used either as a condition of the selection, or the column to select in the generated SQL. The selection API will then produce the correct joins between tables, tables names, and the conditions for the selection from the database. This selection API can be used as a library or as a command line utility.

The command line utility has been named “BbkUser”, and has become quite important to the system for its flexibility. It can provide users with ways of selecting information from the database without the need of knowing table names, the SQL language, or even what a table join is. Users can use the utility to answer detailed questions about the data from the system, without waiting for a developer to provide the feature in some other utility. The developer does not have to guess ahead at all possible combinations of selections that users might need. An example of BbkUser use is shown in Example 4.

Meta-data Distribution

BaBar is a large collaboration, with computing performed at a number of different sites. It was a stated requirement of the bookkeeping system that the meta-data should be accessible by any user at their local site. The local bookkeeping helps a computing site know what data have been imported locally by the data distribution system, in terms of datasets and collections so it is more appropriate to query the local database than the central database at SLAC. This reduces also the load on the central database which is vital for production systems.

To provide for these features, the database can be totally or partially mirrored to any site on demand. The access to each database is granted over the network to all the other remote sites. The meta-data can be accessed by any member of the BaBar collaboration from anywhere.

This creates a series of bookkeeping databases through the world. To simplify the system, each database is used as a read only copy of the master database with is hosted at SLAC. Only the SLAC master database has new

Example 4 : A simple example of the SQL selection API. In this case a user wishes to list run numbers involved in a collection. This is a usual query of the database, and in the selection API it has been simplified to just selecting the value “run” based on the condition of a specific “collection”, as shown in the BbkUser utility. The actual SQL statement is listed after the result in the example, and even though it is a simple enough SQL statement, it requires detailed knowledge of the database, and an understanding of database joins, which is not something that all users should need to know before getting simple queries like this one answered.

```
prompt> BbkUser --collection /store/PRskims/R14/14.4.3d/XiMinus_1550 run
RUN
50488
50489
<...more runs...>
50538
48 rows returned

SELECT bbkr14.bbk_runs.run
  FROM bbkr14.bbk_dsentities,
       bbkr14.bbk_runs,
       bbkr14.bbk_dsetorun
 WHERE bbkr14.bbk_dsetorun.dse_id=bbkr14.bbk_dsentities.id
   AND bbkr14.bbk_runs.id=bbkr14.bbk_dsetorun.run_id
   AND bbkr14.bbk_dsentities.name = '/store/PRskims/R14/14.4.3d/XiMinus_1550';
```

inserts and updates, other databases are synced with the master database.

The requirement to support both Oracle and MySQL produced a problem with the mirroring, since both of these systems had utilities for mirroring from master databases, there was not a utility to mirror between the two systems, so a database mirror application had to be created for the bookkeeping. This mirrors on demand from the central site, and synchronises changes to the database since the last mirror. Each remote site that decides to host a bookkeeping database, can decide how often they need to do the mirror.

The use of a single master in practice has proved not to be a single point of failure, since it will only affect the inserts and updates when it needs to go down. The total down time of the master is small, and new production can easily just wait for updates to the bookkeeping. This does not stop or slow down the production, only the updates to the bookkeeping.

Distributed Connections

Since there will become a number of bookkeeping databases within the collaboration, there was a problem in developing the tools such that they would work no matter which database was in use. The connection information for each database (i.e. server name, user name, and passwords) could not be put into the tools themselves, but a connection API needed to be developed to distribute the connection information on demand.

This was a database connection key distribution system. When a database connection was requested by a utility, the connection keys for that database would be distributed on demand from a central key repository. The definitions of the database along with the connection key would then be passed to the utility.

The system was developed to control access to the use of any relational database within BaBar. The authentication of a user was based on having a unix account at SLAC, and the use of afs and ssh. This was not a limitation within the collaboration, since each member of BaBar is required to have a SLAC unix account. The definitions stored in the key repository can scale for the to be multiple sites in BaBar, multiple servers at each site, and multiple user names within each server, where each can have a different access right.

Using this system users are able to connect to different databases without needing to know any specific information about the servers in use or passwords. Also the tools were developed to work in the same way with different types of databases at different sites. A user can use the same tool with a MySQL database a RAL or an Oracle database a SLAC, and get the same results without event knowing which database was used.

This effect produced a nice feature so there is not a single point of failure in access. Fallback servers can be specified, if the first connection did not work, then the fallback server can be tried. Since the databases are mirrors of the master, read access will give the same results no matter which database is accessed. This fallback happens without the user's knowledge, missing servers will not result in loss of meta-data service.

Distribution of BaBar data

Along with the features to distribute the meta-data databases, the system includes tools to import and export the data in the event store. These import/export tools are driven by the information in the bookkeeping database, and data is distributed in terms of datasets. Users can decide which dataset and component they wish to import (i.e. micro or mini). Importing the dataset and component on demand scales nicely from just a laptop user who needs only one dataset of as little as a few hundred MB, to a large site which wants all of the data (currently 161TB).

Current Status

This system was developed over the past year and a half, and it has provided data access for the last run cycle of BaBar data. The system was beta tested over last fall and winter, and it went into full production in Feb. of this year. The system now contains about 1M runs, 290k files, 184k collections and 17k datasets, and the total size of the database is about 4GB. This is small compared to the total event store of 161TB of data. User feedback so far has been positive.

Task Management

The requirements for the bookkeeping system also includes a request for tools and utilities to help user to keep track of what they process and what files they produce during their analysis. This resulted in a task management system. Where tasks could be specified and then applied to a dataset. This has developed into a large system, which provides tools for the setup and submission of jobs; tracking the output of the jobs; checking the status of jobs and helping in recovery. The system is driven by a database which will keep track of all jobs performed, and task defined. See Ref. [3] for more details.

CONCLUSIONS

BaBar has successfully redesigned its bookkeeping system as required by the new computing model. The core bookkeeping has been created to simplify the users data navigation and selection. It has been designed to give consistent and easy access to the meta-data independently of the production origin. It provides a central point for all the production system to publish the produced collections and for users to access them at any given time. The evolution of the data and in particular of datasets is preserved, allowing users to go back and repeat the same selections and analysis if needed.

The system is distributed to BaBar sites on demand and data distribution tools have been developed to transfer data on demand using the information stored in the bookkeeping to make a user like selection of what a site might want to import. This has been proved to work from a laptop to a Tier-A site. The system will naturally scale to meet the experiments needs.

REFERENCES

[1] P. Elmer, “BaBar computing – From collisions to physics results.” proceedings for CHEP04, 2004.

[2] M. Steinke, P. Elmer, et al., “How to build an event store – The new Kanga Event Store for BaBar.” proceedings of CHEP04, 2004.

[3] W. Roethel, et al. “The BaBar Analysis Task Manager.” proceedings of CHEP04, 2004.