HOW TO BUILD AN EVENT STORE - THE NEW KANGA EVENT STORE FOR BABAR

Matthias Steinke, Ruhr-Universität Bochum, for the BaBar Computing Group

Abstract

In the past year, BaBar has shifted from using Objectivity to using ROOT I/O as the basis for our primary event store. This shift required a total reworking of Kanga, our ROOT-based data storage format. We took advantage of this opportunity to ease the use of the data by supporting multiple access modes that make use of many of the analysis tools available in ROOT.

Specifically, our new event store supports: 1) the pre-existing separated transient and persistent model, 2) a transient based load-on-demand model currently being developed, 3) direct access to persistent data classes in compiled code, 4) fully interactive access to persistent data classes from either the ROOT prompt and via interpreted macros.

We describe key features of Kanga including: 1) the separation and management of transient and persistent representations of data, 2) the modular and extensible persistent event design, 3) the way logical to physical file name translation is done efficiently and 4) BaBar specific extensions to core ROOT classes that we used to preserve the end-user "feel" of ROOT.

BABAR'S EVENT STORE HISTORY

For the first years of data taking BaBar used an event store that was based on the Objectivity OODBMS. Basing the event store on a database technology that was not designed to fit the needs of an event store for BaBar was the origin of a number of problems. Namely

- the concept of concurrent writers is not a parallel approach, which limited the performance of the event reconstruction
- problems concerning with the lock servers such as dead locks were quite common, though transactions are not necessary for the readonly data access of physics analysis jobs
- the system allowed just poor control of the event data to file association. This made the navigation from event data of low level of detail to the corresponding data of higher detail practically impossible if the more detailed data had to be staged back from tape. The tape storage system had to touch too many files.
- the latency for new data becoming available for physics analysis was too large because after the reconstruction the data had to be transfered to the analysis federation first
- the event size was too large because the data was not optimal compressed and there was a too large overhead from the OODBMS
- exporting data to other sites was difficult and resource intensive
- Objectivity is commercial software, and their release policy blocked the migration to new compiler and OS versions.

Many users running physics analysis were not happy with the overall performance of the system, and therefore a second event store which was based on ROOT I/O was developed. This temporary solution allowed a significantly faster access to the so called *Micro* data, the event data of lowest level of detail. It was quite popular for analysis, but it was not usable for reconstruction and MC production.

In 2003 BaBar implemented a new computing and analysis model, refered as *Computing Model 2* [1] [2]. One key component of the new computing model is a new, full featured, ROOT I/O based event store that should overcome the disadvantages of the old solutions, and which in addition should support user customized data and an interactive access to the event data from the ROOT prompt. Our implementation is not based on Grid tools, but follows a more classical approach.

THE NEW KANGA EVENT STORE

Event Design

An event in the Kanga event store consists of *components*, which correspond to the different levels of detail of the event data. A component is implemented as one KanTree, a class derived from TTree, for data and one KanTree for meta data. The clustering of component trees to one or more ROOT files can be configured freely (fig. 1). The maximum file size is limited to 2 GByte. The output module automatically creates a continuation file if the size limit is reached. We try to have file sizes close to 2 GByte to keep the number of files in the event store as small as possible.

References between objects in the same component as well as references between objects in different components, which may be stored in different files, get persisted using the same custom reference class.

The key component of the event is the *event header*. An event header holds a pointer to each of the event components. Such a pointer basically consists of the logical file name of the ROOT file containing the corresponding component tree, and the number of the entry in that tree that corresponds to the event. The component tree can be in the

same ROOT file as the event header, in another file of the same *event collection*, or in a file of another collection .

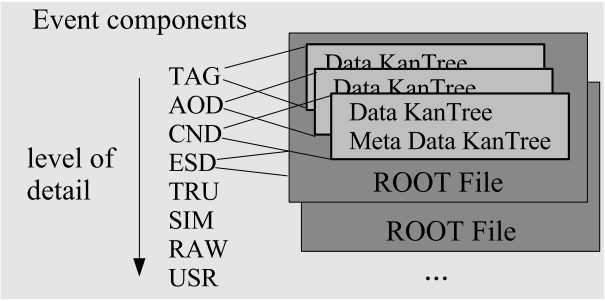

Figure 1: Layout of an event in the new Kanga event store.

Event Collections

Events are organized in event collections, ordered lists of event headers. An event collection can

- own a component: there's a deep copy of the component data in one of the ROOT files of this collection; the component pointer in the event header points to a file of this collection.
- borrow a component: the component pointer in the header points to a file of a different collection. No component data but just the component pointer is stored in this collection.

This feature allows to create very compact pure pointer collections in one case, and more performant deep copy collections in another case. This variety is heavily used in BaBar's skim production, where 120 skims of events of interest for a group of physics analyses get written. Most of these skims have a deep copy of the Micro data and just pointers to the other components.

Because a collection consists of simple ROOT files and because it is relocatable, data import and export was cut down to simple file copy.

File Access

The event collection is the user interface to retrieve event data from the event store. If a collection has to be opened for input, the logical file name (LFN) of the first file with the event header tree of the collection gets constructed from the collection name by a simple naming convention. E.g. from the collection name

/store/PRskims/B0ToccKFinal_0031

the LFN is constructed by adding an extension:

/store/PRskims/B0ToccKFinal_0031.01.root

The translation from LFN to physical file name (PFN) also is done without any file catalog or centrally managed database. The application reads at startup a site specific data access description file with some simple rules (s. fig. 2):

read /store/SP/ file /nfs/workdisk/skims/*

read /store/PRskims/ xrootd kanolb-a:1094/*

The PFN is build by putting the LFN into a namespace according these rules. In this example collections from the simulation production (SP) are read by normal file access from a nfs disk area, and all *PRskims* are read via xrootd using the *kanolb-a* server. For the example collection the resulting PFN is

root://kanolb-a://store/PRskims/B0ToccKFinal_0031.01.root

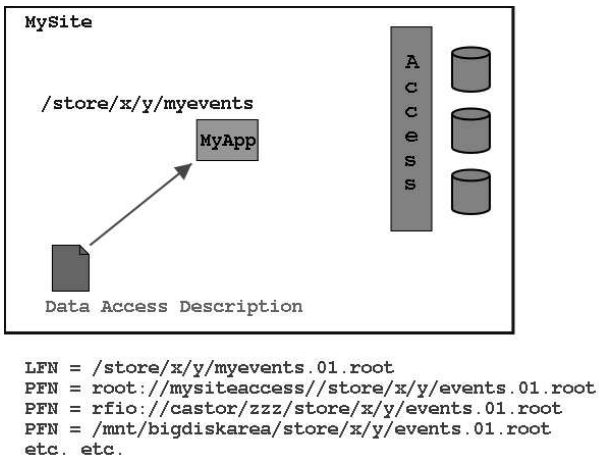

Figure 2: Physical file names get constructed using a simple site specific data access descrition file. No central database is involved.

BaBar's Kanga event store is configured that way that physics analysis jobs always read their event collections via xrootd. xrootd allows a fault tolerant and load shared access to the event store [4].

USR Data and Interactive Access

Two new features that we implemented with the computing model 2 are an event component for user customizable data, and an interactive access to the data in the event store. This allows us to replace most of the n-tuple production and n-tuple based analysis. In the large n-tuples, that were produced by the analysis working groups in the past, a lot of data gets duplicated, and there's no link from n-tuple data back to the event data.

In a skim job the user can define data that gets stored in the USR event component. These data can be associated to the event or to *particle candidates* that are stored in the event. In a subsequent analysis job the USR data can be used like n-tuple data.

The problem that had to be solved for an interactive access to event store data from the ROOT prompt is that in the BaBar computing model transient and persistent objects are strictly seperated and an analysis job runs on transient objects, whereas a ROOT analysis runs on persistent objects. We solved this by adding some functionality to the persistent classes that offers accessors to the transient representations of the data, triggers the unpacking of the persisted data, and caches the transient representations.

Example session The following lines show an example for an interactive Kanga session:

```
root[0] gSystem->Load(libKanga.so);
```

This loads all the symbols that are necessary to use the Kanga data.

```
root[1] KanEventSource* input =
  KanEventSource::mini();
```

This builds an object to read the events (TChain).

```
root[2] input->Add(/a/collection/of/events);
root[3] input->
  Add(/an/other/collection/of/events);
```

This opens some collections for input. Collections are accessed in the same way as in framework applications using the data access description file to construct physical file names.

```
root[4] input->Draw(Pid_DchPids.dedx());
```

This is the simplest possible use, plotting a trivial member function of a stored object. Namely the energy loss of charged particles in the Drift Chamber.

It's also easy to use Kanga with RooFitCore:

```
root[5] TH1FmesHist(mes,mes,100,5.2,5.3);
root[6] input->Project(mes,BSemiExcl_mES);
root[7] gSystem->Load(libRooFitBabar_root.so);
root[8] FitBMass(mesHist);
```

TOOLS

In addition to the core software we deployed a number of tools for administration and maintenance of the event store. The most important tool set are the *bookkeeping tools* [3], which keep track of the event store content and offers an user interface to query the run data base.

To get event collections of convenient size we use a tool to merge the output collections of production jobs running in parallel before they go to the event store.

To make the data stored in the old Objectivity event store available for CM2 analysis we made a conversion application.

There are a number of tools to inspect, move, and delete event collections.

CURRENT STATUS OF THE KANGA EVENT STORE

Currently BaBar's Kanga event store contains ca. 3×10^9 real data events and 3.2×10^9 MC events in 184×10^3 event collections stored in 290×10^3 files. The total size is 161 TByte. All *Micro* and *Mini* data of the run periods *Run1* to *Run3* and the last big MC production were converted from Objectivity to Kanga. During the last run period the *Prompt Reconstruction* jobs wrote directly Kanga event store output, and in the *Simulation Production* the new event store is in use since Feb. 2004. Physics analysis is moving to CM2 since Jan. 2004, and today most analysis jobs read events from the Kanga event store. At SLAC as the largest analysis site in BaBar, usually 2000 analysis jobs are accessing the Kanga event store in parallel.

After moving from Objectivity to Kanga reconstruction and MC production became significantly more stable, mainly because there are no OODBMS related problems like e.g. dead locks anymore. Due to the improved compression and less overhead the event size shrinked, and reading event data got noticeable faster. The latency for data becoming available for analysis after they were written to disk by the DAQ could be reduced from 7 days to 2 days. E.g., after the last data taking period ended, it took 6 days to calibrate, reconstruct, screen and skim all data, and 11 days later the first paper based on these data was submitted to PRL.

BaBar's new Kanga event store, which is a key component of BaBar's new Computing Model 2, is used successfully in production since Feb. 2004, meeting all requirements from data production and analysis.

REFERENCES

[1] P. Elmer, "BaBar computing - From collisions to physics results", presented at this conference.

[2] D. Brown, "The new BaBar Analysis Computing Model", presented at this conference.

[3] D. Smith, "BaBar Book Keeping project - a distributed metadata catalog of the BaBar event store.", presented at this conference.

[4] A. Hanushevsky, "The Next Generation Root File Server", presented at this conference.