SLAC–PUB–7972
October, 1998

Databases for BaBar Datastream Calibrations and Prompt Reconstruction Processes*

J. Bartelt
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

C. Chang
Physics Department, Stanford University, Stanford, CA 94309

S. Dasu
Physics Department, University of Wisconsin, Madison, WI 53706

T. Glanzman and T. J. Pavel
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

Abstract

We describe the design of databases used for performing datastream calibrations in the *BaBar* experiment, involving data accumulated on multiple processors and possibly over several blocks of events ("ConsBlocks"). The database for tracking the history and status of the ConsBlocks, along with similar databases needed by "Prompt Reconstuction" are also described.

Presented at the International Conference on Computing in High-Energy Physics (CHEP98), Chicago, IL, August 30, to September 4, 1998

*) Work supported by Department of Energy contracts DE–AC03–76SF00515 and DE–FG02–95ER40896.

INTRODUCTION

The primary goal of the BaBar experiment is to measure CP violation in the B meson system [1]. The detector is currently finishing construction at the Stanford Linear Accelerator Center, in preparation for cosmic ray running in November, 1998. This will be the initial test of the hardware and software. Beam data-taking, using the new PEP-II asymmetric B-Factory, is scheduled to begin in April, 1999.

The design luminosity of PEP-II is 3×10^{33} cm^{-2}s^{-1} operating at a center-or-mass energy of 10.58 GeV. The BaBar event rate will be 100 Hz, with a raw event size of 32 kbytes. BaBar will use an *Objectivity* object-oriented database "federation" for its Event Store, and Conditions, Configuration and other databases [2–4].

The goal of BaBar "Prompt Reconstruction" (or "PromptReco") is to reconstruct every event in near real-time, within two hours of the data's collection. It will run asynchronously with data-taking, on multiple nodes. The BaBar Online Event Processing software will produce intermediate disk files, called ConsBlocks. Each will be no more than 30 minutes of data. PromptReco will read in these ConsBlocks, processing them with the offline reconstruction code, in a "Prompt Reconstruction Framework" (PRF). This framework process will also log the events to the Event Store [2].

Another task for PromptReco is to accumulate data (histograms, scalars, *etc.*) which the detector subsystems will use for some of their calibration and alignment needs. When a 30-minute block of data (a ConsBlock) has finished processing these accumulations must be summed over the various processing nodes, and possibly over several past ConsBlocks. These summed accumulations are then used to derive constants which are stored in the conditions database and used in reconstructing future ConsBlocks.

Some of PromptReco's database needs are common to other subsystems and processes within BaBar, such as Conditions, Configuration, and Occurrence-logging. Others, such as datastream calibration and ConsBlock databases are specialized, and will be described below.

For more information about Prompt Reconstruction, see: [5,6].

DATASTREAM CALIBRATION AND ALIGNMENT DATABASES

Running within the Framework, each susbsystem will have "modules" to perform reconstruction, monitoring, *etc.*These modules, in some cases, will also accumulate data needed for calculating constants. An example is drift chamber residuals which will be used for determining the constants for the time-to-distance relation. Calibration accumulations of this sort are called "CalChans" [7]. The CalChans accumulated on all the processing nodes for a particular ConsBlock must be summed. These summed CalChans must also be stored for subsystems that need more than 30 minutes of data for their calibration.

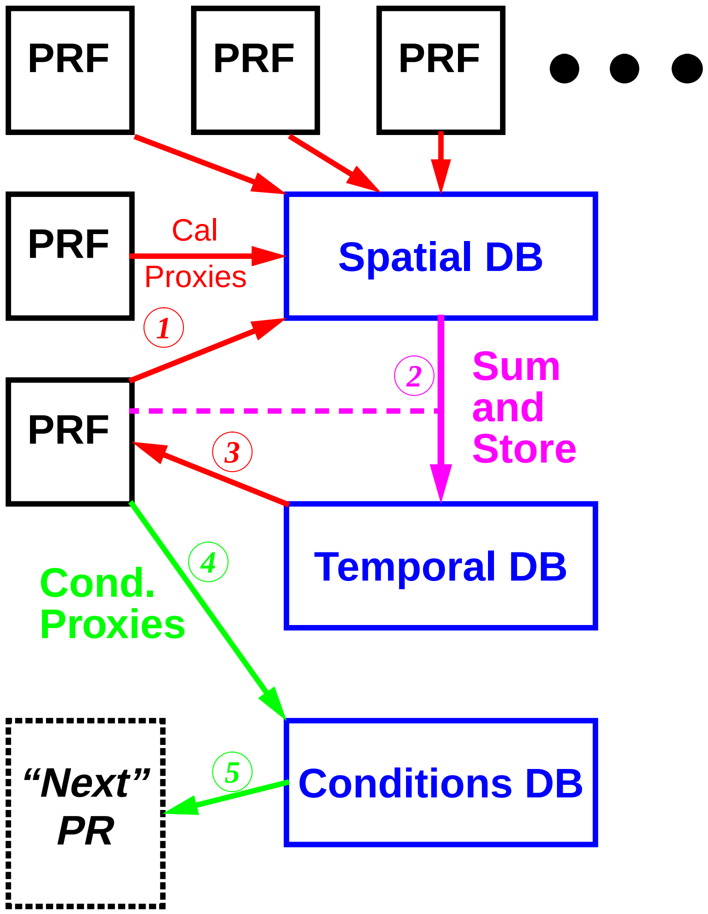

FIGURE 1. Interacting processes and databases for datastream calibrations. See the text for more detailed explanation.

To meet these needs, our conceptual design consists of: (1) a "Spatial" database to hold the CalChans from the individual processors; (2) a "Temporal"database to hold the CalChans that have been summed over all processors for each particular ConsBlock; and (3) a proxy which serves as the user interface to the Spatial and Temporal databases.

The pieces fit together as follows (see Figure 1). As each Prompt Reconstruction Framework finishes processing its events, its modules use a calibration proxy to store their CalChans in the Spatial database (*1*). All but one of the PRF's then exit. The calibration proxies of the remaining PRF sum the appropriate CalChans in the Spatial database and store the sums in the Temporal database (*2*). When this step is complete, this instance of the Spatial database will be erased.

Next, the subsystem code in the remaining PRF requests (via the proxy) the CalChans from the Temporal database (*3*). This request may be for just the last ConsBlock's data, or for the last n ConsBlocks' worth of data. The proxy sums over the CalChans stored in the Temporal database (if necessary) and returns the data to the module. Then the subsystem performs the necessary calculations, and stores the derived constants in the Conditions database [3] via a Conditions proxy (*4*). These constants may then be used in the next (or next+1) instance of PromptReco (*5*).

We also plan to support a request in step *3* for a particular period of time, since ConsBlocks are not guaranteed to be a full 30 minutes long. Such a request would be fulfilled by rounding back in time to the nearest ConsBlock boundary and summing over the appropriate ConsBlocks. If a request for a period of time or number of ConsBlocks cannot be fulfilled, the proxy will return that information. In the future, we also hope to allow for requests which would specify a total integrated luminosity, or some related quantity such as total number of events.

The Spatial database has a hierarical structure based on the "TreeNode" class developed for the *BaBar* Event Store [2]. Each TreeNode can have children which are themselves TreeNodes or "leaves" (other persistent classes). Each TreeNode has a name, and the structure can be navigated like a Unix file system. An iterator class has also been developed. In the Spatial database, there is a single persistent root node. Each instance of PromptReco will have an "Instance TreeNode" which is a child of this root. Each PromptReco Framework processor will have a "Processor TreeNode" which is a child of the Instance Node. These Processor TreeNodes will then have the CalChans as leaves. Each Processor TreeNode will be in a separate container, so that each PRF can store its CalChans without *Objectivity* lock conflicts.

The Temporal database is much like the *BaBar* Conditions database [3], but without versioning. It is indexed by time intervals corresponding to the timespan when the data was collected. It will always contain the CalChans for the most recently processed ConsBlock, and may retain some CalChans for many ConsBlocks. The CalChans are stored and retrieved by calibration proxies.

The proxy used for these datastream calibrations (and for alignment, *etc.*) is the user interface to the databases. It is a templated class which the user instantiates. The proxy mediates the translation between transient and persistent objects, effectively hiding the Spatial and Temporal databases from the user. It is similar to other *BaBar* database proxies.

CONSBLOCK DATABASE

In order to track the history and status of each of the ConsBlocks (the intermediate disk files written by the Online Event Processing, which are the input to PromptReco), we have developed a database for them. A record with the relevant data for each ConsBlock will be stored in the Temporal database. These records can also be stored in an ASCII file independent of *Objectivity.* This file can serve as a buffer to storing the data in the *Objectivity* database, and also supports queries.

Work is underway to generalize this database by templatizing the main classes.

This would allow the same code to be reused for other databases needed by PromptReco: for example, keeping track of CPU and disk resources.

CONCLUSIONS: STATUS AND SUMMARY

Implementation of the major pieces needed for the datastream calibration (proxy, Spatial and Temporal databases) is nearly complete. The work is primarily now to integrate them (and debug them). We have a working ConsBlock database, which will have a better *Objectivity* implementation soon. We expect to have all the necessary functionality for the cosmic ray running in November, 1998.

No performance tests have been done on any of these databases, but we expect their impact to be small compared to the event data. Using the *Objectivity* federated database incurs some overheads, not the least of which is the learning curve for developers. However, using it for these PromptReco databases allowed us to easily integrate them with other *BaBar* databases, and to reuse much of the work that had already been done for the Event Store, Conditions database, *etc.*

ACKNOWLEDGEMENTS

This work was supported by Department of Energy contracts DE–AC03–76SF00515 (SLAC) and DE–FG02–95ER40896 (University of Wisconsin).

REFERENCES

1. *BaBar*Technical Design Report, SLAC-R-457, 1995 (also online at http://www.slac.stanford.edu/BFROOT/doc/TDR).
2. Quarrie, D., *et al.*, paper #33, these *Proceedings.*
3. Brown, D. N., *et al.*, paper #75, these *Proceedings.*
4. Kolomensky, Yu., *et al.*, paper #79, these *Proceedings.*
5. Glanzman, T., *et al.*, paper #52, these *Proceedings*; and references therein.
6. Dasu, S., *et al.*, paper #74, these *Proceedings.*
7. Brown, D. N., *et al.*, paper #76, these *Proceedings.*