SLAC-PUB-9878
July 2003

The BaBar Mini

David N. Brown, representing the BaBar Collaboration
Lawrence Berkeley National Lab, USA

BaBar has recently deployed a new event data format referred to as the Mini. The Mini uses efficient packing and aggressive noise suppression to represent the average reconstructed BaBar event in under 7 KBytes. The Mini packs detector information into simple transient data objects, which are then aggregated into roughly 10 composite persistent objects per event. The Mini currently uses Objectivity persistence, and it is being ported to use Root persistence. The Mini contains enough information to support detailed detector studies, while remaining small and fast enough to be used directly in physics analysis. Mini output is customizable, allowing users to both truncate unnecessary content or add content, depending on their needs. The Mini has now replaced three older formats as the primary output of BaBar event reconstruction. A reduced form of the Mini will soon replace the physics analysis format as well, giving BaBar a single, flexible event data format covering all its needs.

1. The BaBar Experiment

BaBar is a multi-purpose detector operating at the *Pep* 2 asymmetric B Factory. BaBar has been taking data at and near the $\Upsilon(4S)$ resonance since 1999, and has accumulated roughly 110 fb^{-1} of luminosity to date. BaBar is fairly typical of modern High Energy Physics apparati, consisting of several quasi-independent detector subsystems arranged roughly concentricly about the e^+e^- interaction point. The innermost subsystem is the Silicon Vertex Tracker (*Svt*), with roughly 150K readout channels. Outside of the *Svt* is the Drift Chamber (*Dch*), with roughly 7K readout channels. Outside the *Dch* is the Cherenkov Detector (*Drc*), with roughly 11K readout channels. Outside the *Drc* is an Electromagnetic Calorimeter (*Emc*), with roughly 7K readout channels. Outside the *Emc* is the Instrumented Flux Return (*Ifr*), with roughly 60K readout channels.

BaBar was an early adopter of C++ and OO programming in HEP, and the vast majority of our software is written in C++ [1]. BaBar has used Objectivity as the primary technology for storing event data [2], however we are planning to change to a Root based event store by the end of 2003 [3].

2. BaBar Event Data Format History

BaBar's original software design [4] proposed several complimentary event data formats, as described in table 2. These formats were intended to satisfy different use cases, from quality control to reconstruction to calibration to physics analysis, with each format optimized for some specific purposes. Each format was written to separate Objectivity databases (files), so that they could be accessed and managed independently. The *Raw* format is an Objectivity transcription of the raw data readout by the detector online system, and was intended to be used as the input to the reconstruction chain. The *Rec* format represents the reconstructed physics objects, and was intended to be used for detector studies, for detailed analysis, and for single event display. The *Esd* format is a summary of the reconstruction results, and was intended to be the primary format used for high-statistics physics analysis. The *Aod* format was intended to store highly processed information specific to physics analysis. The *Tag* format was intended to store booleans to index and quickly select events. The (*Hdr*) format allows events to 'borrow' some subsystem data from other events, and was intended to support partial re-processing of individual subsystems.

Format	Design Size	Actual Size	Usage
Raw	25 KBytes	50 KBytes	Unused
Rec	100 KBytes	120 KBytes	Unused
Esd	10 KBytes	7 KBytes	Unused
Aod	1 KBytes	3 KBytes	Analysis
Tag	100 Bytes	1 KByte	Selection
Hdr	0	4 KBytes	Unused*

Table I BaBar Objectivity event data formats circa 2001. *Raw* refers to raw data, *Rec* to reconstructed data, *Esd* to event summary data, *Aod* to analysis data, *Tag* to event selection data, and *Hdr* to event header data. The basic features of the *Hdr* format were used routinely, but not the advanced features which made it so large.

By 2001, these data formats and their usage in BaBar had stabilized. As shown in table 2, many of the data formats were not actually used. Additionally, the *Aod* and *Tag* formats were considerably larger than foreseen, and had taken on different roles than originally intended. Subsequent sections explain why the formats were not used according to the original design, and how that led to the development of the Mini.

Work supported in part by the Department of Energy contract DE-AC03-76SF00515.
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309
Presented at the 2003 Conference on Computing in High Energy and Nuclear Physics,
La Jolla, CA, 3/24/2003 - 3/28/2003

2.1. The BaBar Persistence Design

BaBar's original persistence design can be summarized as translating transient objects and transient object relationships into equivalent persistent objects and persistent references, as illustrated in figure 1 for the specific case of reconstructed tracks. The persistent objects were clustered into the various databases according to how it was anticipated they would be used. This design established the now-standard *transient* $\rightarrow$ *persistent* $\rightarrow$ *transient* paradigm in a straightforward way. This design allowed analysis jobs running on (*Esd*) data to retrieve reconstruction details about objects on demand, by following a link back into the *Rec* database. This was considered an important example of how an OO database event store might provide significant new functionality compared to sequentially organized data storage technologies.

The literal translation of complex transient object trees to persistent object trees resulted in a fragmented structure, where different parts of a single physics object (a track in figure 1) were distributed across several databases. This effectively coupled the data formats and database files. For instance, a job reading tracks from *Esd* depended on the *Rec* database to provide the top level tracking persistent object. This coupling added enormously to the IO burden and the disk footprint of an analysis job running on *Esd*.

Similarly, the *Rec* format design required that transient objects be rebuilt from their constituent *Raw* data. This coupled the *Rec* format to the *Raw*, and required that a job reading *Rec* data pull in essentially the entire reconstruction code base. A *Rec* job thus consumed a similar amount of resources (cpu, memory, and disk) as the original reconstruction job.

An additional difficulty to accessing the *Rec* format was that the large size of the *Rec* databases precluded storing them on disk. Instead, they were accessible only through staging. As the staging space at SLAC was originally very limited, dynamic staging through the Objectivity HPSS interface was disabled, forcing users to stage *Raw* and *Rec* databases by hand. This tedious and error-prone operation proved impractical for the vast majority of BaBar physicists.

BaBar originally considered having the online system write raw event data directly in Objectivity *Raw* format. However, since OO database technology was new and relatively untested, a more conservative approach was taken, where the online system writes a flat file version of the raw data, which can then be transcribed into Objectivity *Raw* format. It was then found to be more efficient to reconstruct events by directly reading the online raw data. The *Raw* format was thus recast as an output instead of an input of reconstruction. The *Raw* format was used to pass data between the BaBar simulation executable and its reconstruction executable, until 2001. After that, BaBar switched to a monolithic simulation plus reconstruction executable, eliminating the need for intermediate storage.

2.2. The Unused Formats

The poor performance of jobs reading the *Rec* format, coupled with the difficulty of accessing *Rec* and *Raw* data, made them nearly impossible to use for analysis or detector studies. Only a handful of physicists on BaBar ever made use of either of these data formats, and those uses typically involved very small event samples. Instead, most calibration and monitoring tasks run on raw online data, invoking reconstruction code as necessary to build objects, or on the expanded *Aod* format.

The *Esd* format was not finished in time for BaBar's first data, partly because its development was given lower priority compared to developing the *Raw* and *Rec* formats. It was also felt that some experience with the BaBar detector and data analysis was necessary before the *Esd* could be correctly designed to meet the needs of experiment. It was instead foreseen that BaBar's first data would be analyzed using the *Rec* format, from which experience the *Esd* format could be completed. Since BaBar never used *Rec* to analyze first data (see section 2.3), the *Esd* format was never completed and never used.

The capability of events to borrow content from other events via the *Hdr* format was also never used, mostly because BaBar never encountered a situation which required partial re-processing. The problem of managing the dependent event databases this would have created was also never addressed. The *Hdr* format was recently redesigned to eliminate the data borrowing functionality, greatly reducing its size (see [5]).

2.3. The Evolution of *Aod* and *Tag*

Since it was imperative that BaBar start developing its analysis procedures even before first data, and since the formats originally intended to be used for analysis were effectively unusable, BaBar decided to expand the *Aod* format so that it could be used for physics analysis. This was implemented by including a 'four-vector' summary of the reconstruction results, together with 'quality values' to describe some detector-specific details.

The *Aod* format was spectacularly successful in enabling analysis of BaBar's first data, allowing important physics results to be produced in a timely way. The *Aod* has since been BaBar's primary physics analysis format, and it underlies all the physics results published to date. This format does however have several limitations. For one, the only track fit result stored in *Aod* assumes the particle which generated the track was a π (BaBar reconstruction pro-

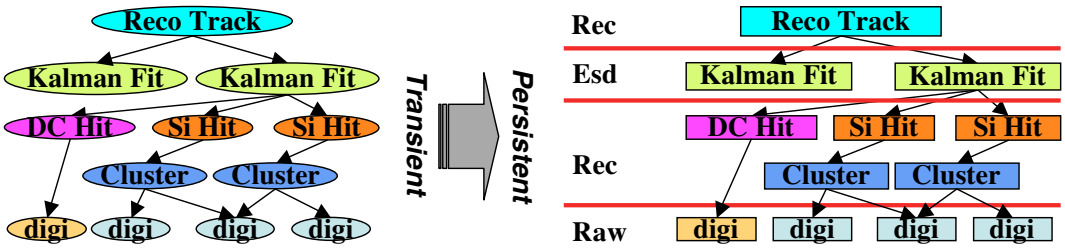

Figure 1: Diagram of the object tree representing a reconstructed track in BaBar, on the left in transient form, and on the right converted to persistent objects according to BaBar's original persistence model. The persistent objects are grouped according to which database they were stored in (the database labels are explained in section 2).

vides all (5) stable particle track fit results). Because the energy loss due to material interactions is mass-dependent, this limitation introduces a small momentum bias when the particle is not a π.

Another limitation of *Aod* is that it reduces the detector information to 'a set of numbers' extracted from reconstruction objects. This greatly reduces the benefits which BaBar might have been obtained from from using OO design and interfaces in analysis. It also effectively isolated BaBar analysis from reconstruction, making it impossible to port code developments from one side to the other.

The *Aod* format also provides a rigid persistent representation of an event, with no way to tailor its contents for specific use cases. Consequently, most BaBar analyses operate by dumping the *Aod* format into an ntuple, adding data content according to thier needs. This effectively doubles BaBar's analysis data storage needs. It also decouples analyses from each other, as different analysis working groups have developed different ntuple representations of the *Aod* format. The redundency and inefficiency of this analysis methodolgy was a strong motivation for BaBar's new computing model, described in section 8.

The *Tag* format also evolved when confronted with first data. To provide more flexibility when selecting events, the *Tag* format was expanded to include floating point and integer values as well as booleans. Thus the intent of the data formats was 'pushed down' one level compared with the original design, with the *Aod* taking the role intended originally for *Esd*, and *Tag* taking the role intended for *Aod*.

By contrast with *Raw*, *Rec*, and *Esd*, the *Aod* and *Tag* formats were developed to be completely independent of the reconstruction objects. This avoided the interdependency problems of the unused formats, at the cost of allowing no way to navigate between physics analysis objects and the reconstructed and/or raw data from which they were derived.

2.4. The Data Format Gap

Because of the evolution of BaBar's data formats, a large gap had developed, with no practical way to access information between raw online data and physics 4-vectors. This gap made performing routine functions like calibrations and detector diagnostics difficult and time consuming. The gap also severely limited the ability to study detector effects in physics analysis. This gap also prevented BaBar from developing a usable single event display. The data format gap was first officially recognized in 2000 during an internal review of BaBar computing [6].

The *Svt* provides one example of how the data format gap caused problems. In order to obtain optimal tracking resolution, the positions of the *Svt* wafers (*alignment*) must be derived from the data. The *Svt* alignment procedure needs both low-level data (hits) and high-level data (tracks) to perform this task. Because of the data format gap, it was found that the most efficient way of doing this was to read raw data, and reconstruct the tracks in the alignment job. The alignment job was therefore very slow, and the total procedure had a turnaround time of roughly 1 month. This was found to be longer than the time interval over which the *Svt* wafer positions were stable. The long turnaround time also stifled development of the alignment procedure, as it took too long to test changes. The net result was that the BaBar used a poor alignment of the *Svt* in reconstruct early data, degrading the effective resolution of the detector, and introducing sizeable systematic errors in many physics analysis. *Svt* misalignment caused the dominant systematic error in BaBar's first sin2β publication [7].

2.5. The Origins of the Mini

To solve the *Svt* alignment problem, a new data format was developed for storing tracks. This new format stored tracks together with their hit data in a compact, flexible, and efficient structure. This new track persistence format was used to design a new *Svt* alignment procedure, which reduced the turnaround

time to roughly 1 day, and which produced measurably better physics results. The success of the new track data format and the new alignment procedure inspired a larger effort to develop a new data format for all of BaBar based on the same basic design. This new format was referred to as the Mini.

The Mini development project was officially begun in early 2001. A prototype of the Mini was produced in a complete re-processing of the BaBar data sample started in 2001. The prototype Mini was used to perform many detector studies, and to refine the Mini design. Unfortunately the Mini prototype did not include any information from the *Emc*, and so was not usable for physics analysis. The first complete version of the Mini was released in early 2002. The design and implementation of the Mini is described in the remaining sections.

3. The BaBar Mini Design Goals

The main design goal of the Mini was to persist the results of event reconstruction. To avoid the problems of the *Rec* format (which had the same goal), the Mini was also required to be small ($<$ 10 KBytes per event), self-contained (no references to objects outside the Mini), and fast (support reading at roughly 20 Hz, equivalent to reading the *Aod* format).

Another goal of the Mini design was to provide access to sufficient detector detail to support standard calibration, alignment, diagnostics, and algorithm development. This capability would also allow the Mini to support a detailed single event display.

The Mini was required to be able to follow changes in *Conditions* (alignment and calibration parameters), so that users could benefit from improved parameters without having to wait for data re-processing. This requirement would also allow analysis users to easily propagate calibration and alignment uncertainties to systematic errors in their analysis, by simply re-running with altered parameters.

To maintain compatibility with reconstruction, The Mini was required to provide access through the interfaces of actual reconstruction classes, without any significant loss in accuracy or content compared to the original reconstruction results.

To support specialized use cases, the Mini was designed to allow users to customize the persistent content according to their needs. This feature is relied on heavily in BaBar's new computing model, as a way of reducing the need to dump data into ntuples in order to do analysis.

To leverage BaBar's huge physics analysis code base, the Mini was required to be compatible with the existing analysis framework. Explicitly, the goal was that an average user be able to convert their analysis job to read the Mini instead of *Aod* without changing any physics-related code, and that the results obtained be equivalent (within floating point precision) to those obtained with the original *Aod* format job.

4. Implementation of the Mini

The Mini design goals require both access to the full detector detail through reconstruction interfaces, together with full compatibility and similar performance as the *Aod* (4-vector summary) format for physics analysis. The Mini satisfies these contradictory requirements by storing both high-level objects (tracks, calorimeter clusters, Cherenkov rings, particle ID, etc), and the low-level objects (*Dch* hits, *Emc* crystals, *Drc* phototubes, etc) from which they were made. This results in some redundancy, as some content of the high-level objects can also be extracted from their constituent low-level objects. Redundancy is generally considered a bad idea in data storage, as it can cause consistency problems. It was accepted for the Mini design as it afforded a large (factor of 10) performance improvement when reading high-level objects (see 7 for details), and because the Mini access mechanism includes safeguards against inconsistent data usage (see section 6 for details).

4.1. High-level objects in the Mini

High-level objects in the Mini store the set of references to the low-level objects from which they were built, thus preserving the essential information of the pattern-recognition algorithm. High-level objects also store references to the other high-level objects they depend on. For instance, *Drc* rings store a list of *Drc hit* references, plus a reference to the track used to seed and fit the ring.

High-level objects also store the results of cpu-intensive functions which their transient class supports. For instance, track objects store the results of running the Kalman filter fit. Because these functions use the associated low-level objects, these stored results are redundant with the low-level objects themselves. Because these functions were invoked during reconstruction, stored results implicitly depend on the *Conditions* which were used when reconstruction was run. Thus, stored results of high-level objects do not follow changing *Conditions*. To follow new *Conditions*, the stored results cannot be used, and the original functions must be called on rebuilt transient objects. Details of how the Mini can be configured to use (or not) stored function results is described in section 6.

4.2. Low-level objects on the Mini

Where possible, the low-level objects in the Mini store raw detector readout information instead of

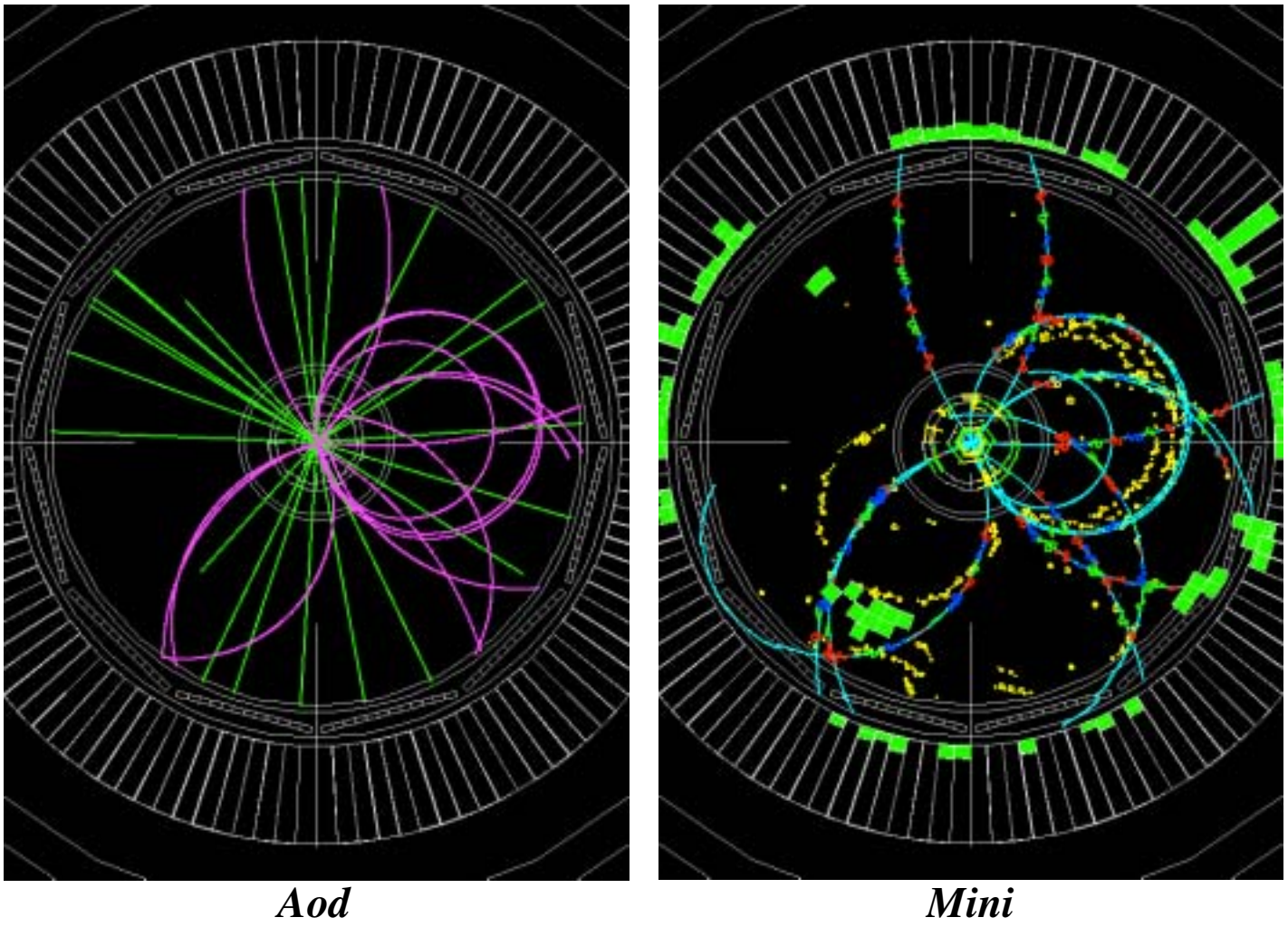

Figure 2: Single event display of a typical BaBar multi-hadron event in *Aod* format on the left, and Mini format on the right. In *Aod*, tracks are modeled as perfect helices, and neutral objects as 4-vectors.

physical quantities. Thus *Dch* hits are stored as TDC values and wire numbers instead of physical times and positions. Physical quantities are then computed from the raw data on the Mini using conversion algorithms and *Conditions* data, as implemented in the transient low-level reconstruction class accessor functions. This allows the Mini to follow *Conditions* changes, and to provide consistent results with reconstruction.

For some subsystems, the raw detector data are very large, and they must be compressed before being stored on the Mini. In these cases, the compressed information is still stored in detector units. For instance, *Svt* hits are compressed to store the average cluster position instead of all the individual strips in a cluster, but the average position is expressed in strip coordinates.

The Mini also stores a subset of low-level objects *not* associated to any high-level object. Monte Carlo and other studies have shown that many of the unassociated low-level objects were generated by particles produced directly or indirectly in the e^+e^- collision. Unassociated low-level objects can be used to identify physics objects missed due to reconstruction inefficiency, or to search for unusual physics signals not found by the standard reconstruction. Associated and unassociated low-level objects can also be combined to create a 'complete' set of low-level objects. This allows the Mini to be used to develop and test pattern recognition algorithms, or to be used as a source for partial re-processing.

Unfortunately, most unassociated low-level objects in a typical BaBar event do not come from the e^+e^- collision. Storing all of them would therefore bloat the Mini and degrade its contents. Instead, only unasso-

ciated low-level objects which pass stringent quality cuts are stored on the Mini. For instance, only those unassociated *Svt* hits whose arrival time is consistent with the reconstructed event time are stored on the Mini. This cut reduces the number of unassociated *Svt* hits by roughly a factor of 20, while keeping roughly 90% of the 'real' unassociated *Svt* hits.

5. Mini Persistence

The Mini was first implemented using Objectivity persistence, and it has recently been ported to use Root persistence, as part of BaBar's new computing model (see section 8). A large part of the Mini's success was due to strict adherence to a few basic persistence design principles, described in the following sections.

The Mini persistence is controlled by the standard BaBar persistence mechanism. A dedicated *loader* module is run for each detector subsystem, which creates the *scribes* responsible for translating specific transient objects into their persistent counterpart. The event key by which a scribe identifies its transient object is configurable through the loader module Tcl interface. Thus the user can control the content of the Mini by choosing which loader modules to run, which scribes to create, and which transient objects the scribes should convert, configurable through control scripts on standard executables.

The configurability of Mini persistence was used to improve the efficiency of the *Svt* alignment procedure (see section 2.5). By reading a custom reduced Mini holding just selected *Svt* track information, the iterative part of the alignment procedure was sped up from several hours to just 10 minutes per iteration. As convergence required hundreds of iterations, this speedup was essential for producing the 23 different *Svt* alignment sets used in the 2002 data re-processing.

5.1. The Persistent Composite Design

The Mini persistence design is based on the *persistent composite* design pattern, in which a single persistent object holds the data for a collection of transient objects of a given type. Using this design, the Mini stores the contents of a BaBar event in just 11 persistent objects, minimizing the impact of the 12 bytes per persistent object Objectivity overhead. A graphical representation of the Mini persistence design is shown in figure 3.

In the persistent composite design pattern, the contents of a transient object collection are stored in persistent arrays (ooVArrays for Objectivity) of *embedded* objects, which have a one $\leftrightarrow$ one relationship with transient objects.

Embedded classes translate to and from their transient counterparts, but otherwise provide no interface. They are implemented as simple structs of primitive data types, with no dependence on any persistence technology. Because they are persistence-free, the same embedded classes can be used by different persistence mechanisms, making it easy to port persistent composite classes other persistence technologies.

Associations between objects on the Mini are stored in the persistent composite objects as a single reference (OORef for Objectivity) to other persistent composites. Specific objects in other composites are then referenced as the *index* into the corresponding embedded array. This results in much less overhead than storing explicit references, as an index (typically 2 bytes) is much smaller than an OORef (12 bytes).

5.2. Data Packing on the Mini

To minimize the size of the Mini, its data contents are packed, according to the following rules.

- Boolean data are stored as a single bit.
- Integer data are stored using as many bits as required by their range.
- Float data are packed and stored as integers. The *Least Significant Bit* of the packed data (LSB) corresponds to roughly 1% of the intrinsic detector resolution of the quantity being stored. Float data with an extreme natural range are packed logarithmically, using an algorithm which is locally flat to avoid binning artifacts in histograms (see figure 4).
- Packed integer and float values are truncated at physically reasonable ranges, not 'worst possible' ranges. Values beyond the physically reasonable range are flagged as under or overflows.
- Strings are stored as a key (integer) in a string $\leftrightarrow$ integer map. The map is stored outside the Mini event data.
- Small data fields are combined (bitwise OR) to fill a standard type (char, short, or long) word.
- Data members of embedded classes are all aligned to either char, short, or long word boundaries (one choice per class), to ensure that embedded object arrays are compact in memory.
- To avoid the Objectivity overhead of storing and retrieving virtual tables, embedded classes have no virtual functions, *including no virtual destructor*.
- Direct data members of persistent composite classes are aligned to long word boundaries, to

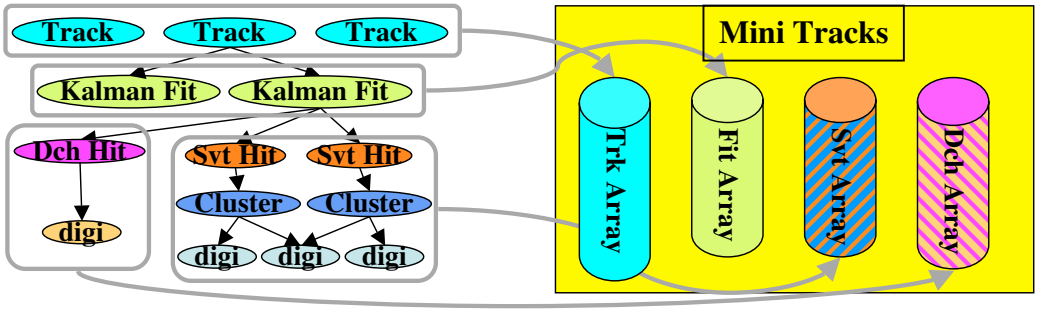

Figure 3: Diagram of the object tree representing a reconstructed track in BaBar, on the left in transient form, and on the right converted to Mini persistent format. The Mini stores all tracks in an event in a single persistent object. Track-specific details are described by the embedded objects stored in the different persistent arrays.

be consistent with Objectivity persistent object alignment

- To avoid creating persistent memory fragments, variable arrays (ooVArrays) are sized exactly once, either on initialization or in the constructor body.

6. Accessing Mini Data

As described in the previous section, the Mini persists several levels of data which overlap in content. The Mini is designed so that the user must decide which level of detail is appropriate when reading back the Mini. In making this decision, the user must balance the greater detail and (potentially) greater accuracty which are available when reading low-level data, against the greater speed possible when reading high-level data. The Mini persistence provides a very precise degree of access control, so that some objects may be read with high precision while others are read with lower precision. Similarly, it is possible to read an event initially at low precision, and later upgrade some or all objects to higher precision once the event or the objects in it pass cuts. Maintaining coherent and correct data contents under these general conditions is however difficult, and involves a level of expertise beyond that of the average user.

To make it easier for users to correctly configure reading the Mini, a set of self-consistent access modes are provided which roughly span the available options. While some users may need to optimize their Mini jobs by directly controling the persistent access, it is expected that most Mini users will choose one of these five access modes: *micro*, *cache*, *extend*, *refit*, and *raw*. The access mode is set in a user job through the *levelOfDetail* global Tcl variable, which is then used by the sequences which read and prepare the Mini data for analysis. The specifics of these different modes are described below. A comparison of the access speed in these different modes is given in table III.

The Mini user can also control how the BaBar *Conditions Database* [8] is accessed. For instance, a user can configure their Mini job to use the same *Conditions* as were used when the data were originally processed, or the most recent *Conditions*, or even to override the *Conditions Database* and use explicitly-provided constants. *Conditions* access configuration is most relevant to refit and raw modes.

6.1. Cache Mode

Cache mode refers to reading the Mini so that high-level objects are built from the stored function results instead of low-level data. Only a summary of the low-level information can be obtained in cache mode. For instance, in cache mode, the track transient object can provide the number and logical identity of the hits from which it was fit, but it cannot produce the actual hit objects. Since a cache mode job doesn't read or process any low-level data, it is much faster than a refit job run on the same data.

In cache mode, all the stored track fits can be used. The default version of the mini stores all the unique mass hypothesis Kalman fits, evaluated at their point of closest approach to the Z axis, plus the π fit evaluated where the track exits the tracking volume.

6.2. Micro Mode

Micro mode is a variant of cache mode where some features are turned off, in order to make the Mini behave more like the *Aod* format. For instance, since *Aod* stores only π track fits, in micro mode the Mini track provides only the π fit. Micro mode is intended to make it easy to compare and validate the Mini against the *Aod* format. Since micro mode is no faster than cache mode, and yet returns less accurate values, it is not recommended for use in physics analysis.

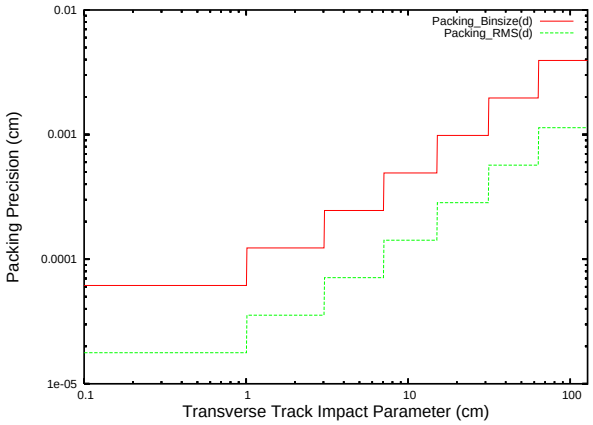

Figure 4: Resolution of the globally-logarithmic, locally flat packing algorithm used in the Mini, as applied to packing the track transverse impact parameter. This quantity has a detector resolution of roughly 10 microns, and range of values from 0 to 80 cm. The packing algorithm employed is extremely efficient to unpack, involving only 2 floating point operations (one addition and one multiplication).

6.3. Extend Mode

Extend mode is a variant of cache mode in which the validity range of a track is extended from the range of the fit result stored on the Mini, up to the first hit. Otherwise, extend mode behaves exactly as cache mode. The persistent data read in extend mode are exactly the same as in cache mode, but since more tracking functions are called, extend is somewhat slower to read than cache.

In extend mode, the fit results stored on the Mini are interpreted as a multi-dimensional 'hit', constraining all the track parameters to the stored fit values. These 'constraint hits' are used to create a Kalman track fit object, which is an instance of the same Kalman fit class used to fit tracks in BaBar reconstruction. The Kalman fit adds the effects of passive material and magnetic field distortions as the track traverses the the detector, extending the range over which the stored track can provide physically accurate parameters. Since hits are not read in extend mode, extended tracks are valid from the origin out to the first hit.

An example where extend mode is useful is reconstructing long-lived particles which decay outside the beampipe, such as K^0. In cache mode (and when using the *Aod* format), these particles are vertexed using fit results measured inside the beampipe. In extend mode, track fit results are measured at the decay vertex, so that the reconstructed parameters of the K^0 are more accurate and less biased.

6.4. Refit Mode

In refit mode, the function results stored with the high-level objects are ignored, and high-level transient objects are rebuilt from constituent low-level objects. Because refit mode involves reading more data and performing more computation to create the high-level objects, it is substantially slower than either cache or extend mode. Because new *Conditions* are read and used when rebuilding the transient objects, a refit mode job can follow changing *Conditions*, or even changes in some reconstruction algorithms. Refit mode is intended to support detector studies, single event display, specialized analyses that depend on low-level data, and analyses that need to use new *Conditions* or algorithms.

6.5. Raw Mode

In raw mode, high-level objects stored on the Mini are ignored. Both assigned and unassigned low-level objects are are read and combined together into 'complete' lists, and the reconstruction pattern recognition algorithms are run on those. Raw mode is intended to support development of pattern-recognition reconstruction algorithms, to support re-processing, and to support event mixing studies. Because raw mode invokes pattern recognition algorithms, it is slower than refit mode.

While the high-level objects created when reading the Mini in raw mode are similar to those read in the other modes, they are not necessarily identical, as the initial sets of low-level objects are not exactly the same as those used when running reconstruction on raw online data.

Raw mode is still under development as a user option, though it has been tested in a limited form.

7. Performance of the Mini

General performance numbers for the Mini, such as size on disk and read speed under various conditions

Data	Generic	Multi-hadron
Mini	6.4 KBytes	10.0 KBytes
analysis reduced Mini	1.8 KBytes	3.2 KBytes
Aod	1.8 KBytes	2.7 KBytes

Table II The average (compressed) size of BaBar events stored in Mini, analysis reduced Mini and *Aod* formats. Results for the analysis reduced Mini are based on a prototype.

mode	Generic	Multi Hadron
micro	45 Hz	24 Hz
cache	45 Hz	22 Hz
extend	28 Hz	14 Hz
refit	5.3 Hz	2.4 Hz
raw	3.3 Hz *	1.0 Hz*
Aod	246 Hz	173 Hz
analysis reduced Mini	96 Hz	-

Table III The event rate reading the Mini in different modes on an 1.4 GHz Pentium Linux machine. The times for raw mode were estimated using the BaBar reconstruction executable, as this Mini mode has not yet been fully implemented. Results for the analysis reduced Mini are based on a prototype.

are listed in tables III and II. Performance of the *Aod* format is given for comparison. As efforts to optimize the read speed of the Mini have only just begun, these numbers should be considered provisional. Table IV gives a breakdown of where time is currently spent in a typical Mini analysis job. This clearly shows that unpacking data plays a very minor role in the performance.

8. BaBar's New Computing Model

In April 2002 BaBar computing was reviewed by a combined internal and external review board. Among other recommendations, the report of this committee [9] suggested that BaBar reconsider its Analysis Model in light of the opportunities offered by the Mini. In response to these recommendations, a new Computing Model was adopted by the collaboration in December 2002. This model introduces two major changes, first that the BaBar event store be converted to use Root persistence instead of Objectivity, and second that the existing physics analysis format (*Aod*) be replaced with a new format more consistent with the Mini. After some discussion, a reduced Mini customized for analysis has been chosen as the *Aod* format replacement.

Operation	% time
Reconstruction transient creation + deletion	35
Objectivity data read	25
Physics interface adapter	20
Event loop overhead	10
Data unpacking	0.1

Table IV The fraction of time spent in various operations when reading the Mini in cache mode in a standard physics analysis job.

To replace the *Aod* format, the analysis reduced Mini must have a similar size on disk and read-back speed as *Aod*. The starting point for achieving this is to store only those quantities referenced in cache mode. Performance results from a prototype analysis reduced Mini are given in tables III and II, showing that it is similiar to *Aod*. A major effort is now underway at BaBar to improve the read-back performance. Based on profiles of a standard analysis job, the largest time sinks come from inefficiency in the reconstruction code invoked when reading the Mini, and in the analysis interface to the Mini (see table IV). Based on the problems already identified, the read speed is expected in increase by between a factor of 2 and 10.

In the new BaBar Computing Model, the analysis reduced Mini will store only the cache mode information. The remainder of the Mini information will be stored in a separate file. The complete Mini will be accessed by reading both the reduced and remainder information. Thus BaBar will store event data in a coherent format, split into pieces specialized for analysis and detector studies, with no redundancy and easy navigation between the two pieces.

A requirement of the new computing model is that the *Aod* replacement be accessible interactively. As part of satisfying this requirement, BaBar has chosen Root as the persistence technology for the *Aod* replacement, since it has been shown to work as a HEP event store technology both at BaBar and elsewhere, and because the Root/CINT interface is a standard interactive access mechanism. To satsify this requirement, the Mini is being ported to Root persistence. The Mini Root persistent implementation uses base classes developed at BaBar which allow interactive access to packed data contents of embedded objects, by dynmaically linking class functions into Root [10].

A key feature of the new computing model is the ability to create custom output streams for physics groups, by exploiting the configurability of the Mini. Coupled with the interactive access capabilty afforded by Root persistence, it is hoped that custom streams can replace the *Aod* format dump ntuples used in most analyses. This will substantially reduced the computing and human resources used in analysis.

Because BaBar is a functioning experiment, the new computing model must be introduced in a way that does not disrupt ongoing efforts, and quickly enough that its benefits can be exploited before the experiment ends. The plan is to develop and deploy the new computing model within calendar year 2003.

9. Conclusions

BaBar has introduced a new event data format refered to as the Mini. This format addresses deficiencies in BaBar's older formats.

BaBar has just completed a full data re-processing, in which the complete Mini replaced the unused *Raw*, *Rec*, and *Esd* formats. This reduced the volume of data produced in reconstruction by a roughly factor of 10, significantly improving the efficiency of the event reconstruction farm, and requiring half as many data servers compared to previous processings. BaBar's data storage costs were also cut by roughly the same factor of 10.

BaBar is now starting to use the Mini for physics analysis. An ambitious new computing model has been adopted, in which a reduced form of the Mini will replace the current physics analysis format. When the new computing model is deployed in late 2003, BaBar will have a coherent event data format covering most of the needs of the experiment, finally satisfying the intent of the original format design.

Acknowledgments

The authors wish to recognize the achievements of the BaBar offline software developers who designed and implemented the BaBar reconstruction framework and event persistence, which has been spectacularly successful in enabling BaBar to produce high quality physics results, and which laid the foundation for developing the Mini.

References

[1] B. Jacobsen et al, *Applying Object-Oriented Software Engineering at the BaBar Collaboration* Proceedings of CHEP 1997, paper 372.

[2] D. Quarrie et al, *First Experience with the BaBar Event Store* Proceedings of CHEP 1998, contribution 33.

[3] D. Kirkby et al, *KANGA(ROO): Handling the micro-DST of the BaBar experiment with ROOT* Proceedings of CHEP 2030, contribution TUKT007.

[4] D. Quarrie, *The Design of the BaBar Event Store* June 1997 (BaBar computing internal document).

[5] A. Adesanya et al, *The Redesigned BaBar Event Store - Believe the Hype* Proceedings of CHEP 2030, contribution TUKT008.

[6] *Report of the Review Committee on BaBar Computing* September 2000 (BaBar computing internal document).

[7] *Measurement of CP-violating asymmetries in B0 decays to CP eigenstates* Phys. Rev. Lett. **2**001 86 2515-2522.

[8] A. Trunov et al, *Operational aspects of dealing with the large BaBar data set* Proceedings of CHEP 2030, contribution TUKT006.

[9] *Report of the Review Committee on BaBar Computing* April 2002 (BaBar computing internal document).

[10] *Upgrading BaBar's use of Root Persistence to Support Interactive Access* http://www.slac.stanford.edu/ echarles/work/Kanga.