# BaBar and ROOT data storage

Peter Elmer
BaBar −Princeton University

ROOT2002 −14 Oct. 2002

# The BaBar experiment

- BaBar is an experiment built primarily to study B−physics at an asymmetric high luminosity electron−positron collider PEP−II at the Stanford Linear Accelerator Center (SLAC) in California
- BaBar is an international collaboration involving 560 physicists from 76 institutions in 9 countries
- We have been taking data since May, 1999 and are currently finishing preparations for ‘‘Run 3’’ which will begin next month
- To date we have taken ~95 ifb of data, we expect to increase this to o(0.5 iab) by ~2005

## Overview

- In this presentation I am going to talk about the use of ROOT in BaBar for event data storage in particular.
- I’ll discuss the current model for use of data stored in ROOT files as well as extensions to that model being deployed currently
- I will also mention some studies to see what we would need for a full eventstore based on ROOT I/O

## Data Formats in use in BaBar

- L3 trigger writes raw data custom flat file (‘‘xtc’’ file)
- Reconstruction reads xtc file and writes into Objectivity Eventstore
  - ➔In the past we wrote raw, reco, mini, micro, tag
  - ➔Now we write <u>mini, micro, tag</u> (i.e. no bulk)
- Monte Carlo Simulation writes into Objectivity Eventstore
  - ➔ In the past we wrote sim, tru, raw, reco, mini, micro, tag
  - ➔Now we write (mostly) <u>tru, mini, micro, tag</u> (no bulk)
- Dedicated converter application used to produce MC and data in ROOT files (‘‘Kanga’’) for analysis use
  - ➔Primarily <u>micro, tag</u> data
  - ➔Simple format: one file per (data or MC) run

# Analysis usage by site

- The development of the Kanga/ROOT format as an alternative to Objectivity for analysis was primarily intended to support analysis in universities, but has found adherents even at the larger centers. Simple file based model eases data distribution, low maintenance overhead.
- SLAC Tier A −Now primarily Objectivity data, until recently also provided access to data in Kanga/ROOT format
- In2p3 Tier A −Objectivity data access only
- RAL Tier A −Kanga/ROOT data access only
- Karlsruhe (FZK) and Rome Tier B −Kanga/ROOT data access only
- Many university sites (Tier C) −Kanga/ROOT data format

# BaBar data sample size

- Event size (compressed):
  - Data: Micro ~ 2kB/event, Mini ~ 8kB/event
  - MC average: Micro ~ 5kB/event, Mini ~20kB/event
  - Raw ~ 30kB/event (xtc)
  - Tag ~200bytes
- Total raw data set size ~ <u>**100TB**</u> (xtc files)
- Total objectivity data set size ~660TB (includes raw, rec, micro, mini, tag and also sim/tru data components from MC) of which ~<u>**30TB**</u> is (micro,tag)
- Kanga/ROOT data set size (micro/tag) ~<u>**19TB**</u>
- The Kanga data set includes multiple copies of many events due (in part) to creation of multiple streams for export. Objy copy of the dataset has no duplication, but has no compression to hide sins of poor packing as well as large navigational components

## Analysis data access

- Different access modes for the two analysis formats:

➔Objy Micro: analysis access via AMS, mostly disk resident at SLAC, dynamic staging used more agressively at In2p3

➔Kanga micro: Disk resident, access via NFS at all sites

- We use an architecture with a clear transient/persistent split so that user code can run on either format transparently
- Analysis so far has focused mostly on micro data, recent improvements and extensions to the mini data have made it a lot more attractive and useful. Two principal problems: it is a factor of 4−5 larger and has been implemented thus far only for Objectivity.

## Data organization − ROOT files

- Micro/tag data is organized very organized into a single tree, with <u>one tree per file</u>, <u>one file per run</u>. One run for real data is the typically the time between fills, i.e. 1−2 hours, and is typically 200k events. For MC data it is the output of one batch job, typically 2k events.
- Data has a ‘‘logical’’ path and name under a common root directory, soft links are used to map that to physical volumes, a simple script is used to move things around for space and data management reasons.
- User generates list of ‘‘logical’’ file names from a catalog in a relational database using a script and uses this to configure his/her jobs. Running jobs talk only to the data.

## Skims and index collections

- I mentioned in an earlier slide that there was duplication of events in the 19TB of Kanga/ROOT data. This was done primarily to allow export.
- We have an ’’AllEvents’’ stream and in the past have had up to 20 other streams containing subsets with particular physics selections. Any given event was available both in AllEvents as well as one or more of the 20 streams.
- There was a total event duplication factor of ~2.5 between these streams which requires additional disk space (but also optimizes access to sparse samples)
- We are now putting into production custom index collections into a much smaller number of streams (currently 2 physics + 1 calibration) with negligible duplication

## Skims and index collections II

- The idea is that only the main streams will be available at the large centers along with index collections. Smaller centers wanting a particular physics subselection can use the index collections (or tag bits) to deep copy the sample they are interested in.
- Have created an export system consisting of a client script (running at a ‘‘Tier C’’, for example) which:
  - Starts a rootd on one of a range of ports
  - Generates a random password and stores it in a file
  - Connects via ssh to ‘‘Tier A’’ (e.g. RAL)
  - Sends password across ssh connection and starts deep copy executable
  - Deep copy executable contacts rootd and writes output across WAN to Tier C

# General notes and questions

- We were using the very old ROOT 2.23−12 through this spring, now using 3.02−07.
- ROOT is now in the ‘‘required’’ software list in BaBar (BaBar limits to a handful the number of ‘‘external’’ software packages that are needed to use the BaBar software)
- Is there a tool to configure ROOT shared library dependencies based on the actual include files? (root−config −−libs and −−glibs probably wrong granularity for configuring dependencies as we might like, basically ‘‘almost everything’’ and ‘‘everything’’)

## Objectivity contingency plan

- As an exercise in understanding what would need to be done to migrate away from Objectivity should that ever be necessary, we explored how one could extend the Kanga/ROOT system.
- The BaBar framework for writing persistent objects with ROOT is more general than this, allowing objects from different components (micro, mini, sim, tru, ...) to be written to separate trees in separate files (as in the Objectivity event store).
- Some work was needed (and done) to generalize this a bit and to provide full infrastructure to teach BaBar executables to read and write multiple components into root files with that framework.
- o(100) persistent classes needed to be implemented. No obvious way to simply translate objectivity ddl to root classes, but most are fairly simple and were done by brute force. Remaining are o(20) mini classes.
- This portion has taken a bit of work, but is largely straightforward to finish. Recent ROOT support for foreign objects is very interesting in this context, especially for the remaining mini classes.

## Other bits needed

- Transactions, which in practice meaning checkpointing
  - ➔For MC, this is not so important. Each run is just a single batch job. A run which fails before finishing all events is just rerun or dropped. The quantum of cpu lost is not so large when few jobs fail (i.e. in production mode).
  - ➔For real data, we process a run by parallelizing the events across a farm of cpus. We want to process every event once and only once. For this, checkpointing is more useful
- Used TTree::AutoSave() to checkpoint every N events, this maps trivially to existing mechanisms used for Objectivity commits and notification of server distributing events
- Works fine in tests with individual jobs, in particular for the most common case where the job fails during the event processing itself. Need to understand better the parasitic cases where the job fails while ‘‘committing’’.
- Still need to do bulk tests with many clients (soon...)

## Dynamic staging

- Two distinct problems: small file size for ROOT files and staging mechanism itself
- We believe that we can deal with the small file size simply by writing to (and reading from) tape a (tar) archive. That is in fact how we store the ROOT files currently to tape, although we don’t dynamically stage them.
- The principal issue is how to add support for dynamically staging these root files from HPSS (used at SLAC and In2p3, in particular)
- For Objectivity this is done using the AMS, and a daemon based solution also for accessing the ROOT files is also desirable, allowing us to use the same backend for the HPSS access.

# Dynamic staging II

- One possibility might be to extend or augment the rootd as a replacement for the AMS in this role.
- New plug−in architecture should help with this
- There are several things that could be done as part of such an extension that might be of more general interest:
  - ➔Request defer protocol
  - ➔Request redirection protocol
  - ➔A negotiated server−directed security protocol
- Are other groups doing (or looking to do) similar things (with rootd)? Is there similar work in progress or planned (or done) elsewhere?

## Interactive analysis

- I mentioned earlier that we use a transient−persistent split which allows us to support both objectivity and root outputs as well as evolve aspects of event storage independent from the development of analysis, reconstruction and simulation code.
- There is also some interest in being able to do some amount of limited analysis directly in root with the Kanga/ROOT
- Some work has gone into this recently and is ongoing
- Two basic problems in supporting both this and ‘‘framework’’ based analyses:
    - ➔Accessors −information in the persistent representation is often packed, need to add functionality to allow this to be called by the user
    - ➔Navigation −most of the intelligence for navigation is not in the persistent classes themselves, but in the code for doing the persistent (to/from) transient conversion.

## Interactive Analysis II

- Solved by moving more intelligence down into the persistent classes themselves, support both full analysis on transient objects and interactive analysis either directly or via transient to persistent conversion
- This is still evolving and is convolved with two other things we are looking at:
    - ➔A redesign of the micro (cheaper, better, faster)
    - ➔The desire to allow users to augment production data with user information without duplication
- Once the dust settles, it would be useful for me to be able to discuss some of the ideas with the ROOT team

## Conclusions

- The use of ROOT I/O for data storage has and will likely continue to be very useful for BaBar.
- A simple file based model using ROOT files with the catalog in a relational database has proven to be robust, scalable and straightforward to use for sites wanting access to micro data for analysis, but minimal maintenance overhead.
- The way (and extent to which) we are using ROOT for data storage is still evolving...