BaBar Computing Challenges (personal recollections)

Gregory Dubois-Felsmann, SLAC
BaBar Computing Coordinator 2005-yesterday
LSST Data Management system architect from 11/15

B-Factory Symposium - 26 October 2008

BaBar Computing Challenges

- Personal recollections and views of lessons learned, not a comprehensive history

- Road map:
  - The shape of the problem and the size of the challenges
  - Building C++ software for an HEP experiment from scratch
  - Taking data: pursuing online performance and efficiency
  - Code and configuration management under pressure
  - Providing resources: coordinating international contributions
  - Getting the physics done: working with analysts
  - The long haul: beyond construction and commissioning
  - The very long haul: data preservation and wider access

The shape of the problem
and
the size of the challenge

The shape of the problem

- Unprecedented luminosity and challenging backgrounds
- A moderately complex detector (~2×10^5 channels) with electronics based around gigabit fiber readout
- Physics goals requiring "factory mode" operation and minimal systematic uncertainties, thus...
- Excellent trigger efficiency for key physics: >99%, robust to minor detector element failures, with <~1% deadtime
- Prompt reconstruction of all data for quality checks and physics-ready data

The size of the challenge - data acquisition

- At design luminosity (3×10^{33} $cm^{-2}s^{-1}$) , trigger simulations led to requirement for 2,000 triggers per second (Tps) DAQ capacity, with stretch goal of 10 kTps, for 35 kB events
  - Ultimately: 7 kTps during final Y(2S/3S/above-4S) running
- Software trigger to reduce stream to ~120 Tps at design luminosity, for archival logging
  - Ultimately: ~400 Tps during highest-lumi running (~12×10^{33} $cm^{-2}s^{-1}$)
- Challenging throughput
  - ~30 GB/s peak rate off detector (~350 gigabit fibers)
  - 70-250 MB/s of hardware triggers
  - 4-15 MB/s of continuous data logging

The size of the challenge: offline processing

- PEP-II running hours greatly exceeded historical “Snowmass year” norms

- First year of running produced 23 fb^{-1} of data and one billion logged events, 330 million fully reconstructed

1999-2008:

- 531 fb^{-1} and 22 billion colliding-beam events logged, 0.7 PB of raw data
- 8.7 billion events fully reconstructed; similar amount of MC
- Multiple processing and simulation passes, adding up to about 3 PB of processed data (0.7 PB for largest pass)

The size of the challenge: the human element

- *BaBar* was one of the first all-C++ experiments, so almost everyone learned on the job
- "Everyone is a developer" in HEP experiments
- Active collaboration size of ~500 at any given time
- ~250 people made meaningful contributions to *BaBar*-specific reconstruction, simulation; much of this was "collaboration service" work without the leverage of direct administrative supervision
- 100-200 active analyses at any time for most of *BaBar*'s lifetime so far

- Continual focus on competitiveness, constant improvement

The shape of the solution

This year, at the close of data-taking:

- About 3 million lines of offline code, 0.5 million online
- 24 major release cycles so far, hundreds of releases
- 160 embedded 300 MHz PPC CPUs (DAQ) and 200 modern x86 cores for software triggering and data quality monitoring
- 4000-5000 x86 Linux cores at SLAC, similar number elsewhere, for data production and analysis
- ~800 TB disk space (bulk data, production scratch, user space) at SLAC, similar amount elsewhere
- 5-10 TB/day data flows among the major international BaBar sites to support distributed production and analysis
- ~20 *BaBar* and ~20 laboratory computing FTEs at the peak

Results

- More than 96% of all available luminosity was recorded, of which virtually all produced good data for physics
- In time for nearly every year's summer conference cycle, a high-quality dataset was made available to analysts within 10 days of the acquisition of the last data
- The extraordinary physics you will see this afternoon

Building C++ software for an HEP experiment from scratch

The adventure of C++

- No previous large experiment had used C++ in production for more than a fraction of its code base
- Although C++ appeared to be well-accepted in the professional software development community, the experience base in HEP was tiny and largely limited to R&D
- The limitations and problems associated with the dominant Fortran 77 model were becoming very clear, and the core group of physicist-developers very enthusiastic about adopting a more modern model - but there were very serious concerns about its accessibility for the larger community, and about performance
- Yet *BaBar* chose early on to commit to this technology

Motivation and direction

- Why?
  - System engineering considerations (encapsulation, abstraction); strong belief that BaBar would turn out to require more complex software than previous experiments and that old approaches would not scale - C++ was seen to promise a lower life-cycle cost
  - Perception that the HEP data model was a very good match to an object-oriented design and that this was *ipso facto* compelling
  - Core team enthusiasm

- "Mid-1990s" style of C++ design chosen: strongly object-oriented, based on abstraction and polymorphism
  - Few templates, no STL
  - Prototyping was encouraged, some community R&D projects used (CLHEP, track fitters, MC frameworks, event generation)

One more choice

- Very large data volumes were anticipated
  - Expected to make it impossible to routinely run on all the data
  - A successive-refinement model of analysis was envisioned
- Traditional HEP data analysis and persistency mechanisms looked like they would not scale
  - Either in performance or in manageability
- The outside world appeared to have just the solution: object-oriented databases
  - Persistency model matched directly to C++
  - Implicit promise that a database system help would help with managing a very large dataset
  - Promise that a database would naturally aid with event selection

Teaching a collaboration C++

- At first, a few knowledgeable collaboration members started offering C++ tutorials
  - Programming guidelines and design/analysis techniques were harvested from the published literature
- The turning point (starting in 1996):
  - Formal training for a core group of developers (through this year)
  - Commercial consultants were brought in to teach object-oriented analysis and design (and Objectivity/DB), 15 BaBarians at a time
  - The experience of going through this together rapidly produced a shared vocabulary and set of concepts
  - I believe this was *essential* in allowing the project to succeed
- Crucial decision:
  - Aggressively designing for performance was too much to ask of new C++ hands (except in the online)

Results

- It turned out well - as you'll see today! - but it was close
  - Careful nurturing and rethinking was needed all along
- Opinionated scorecard:
  - Online computing and software triggering: unambiguous success
    - Core DAQ and software trigger code was all C++ and ran efficiently and met all performance requirements all along, readily upgraded as needed
  - Reconstruction and simulation algorithms: mixed results
    - Remarkable physics performance achieved with OK maintainability
    - Computational performance clearly suffered from inexperience with C++, and with its more-opaque connection with CPU/memory impacts
    - An investment of strong FTEs in code reviews would have paid off well
  - OO databases: disappointment - not as good a match as anticipated
  - User experience: data model unnecessarily difficult for new users
    - Some physics productivity was probably lost to this
    - I don't think this was inherent to C++; need to digest the lessons

Taking data:
pursuing online performance and efficiency

Maintaining 96%+ for 9 years: the necessary mindset

- Major investments in instrumentation for performance measurement enabled targeted optimization. Result:
  - Quadrupling of luminosity, throughput without loss of efficiency
- "Factory" means reliability and maintainability are key
  - Uncompromising focus on fixing even very small problems as soon as they occur more than once
  - Sustained pursuit of improvement in operability measures
    - E.g., shutdown/restart times, recovery from error conditions
    - Multiplier effect, because these speed debugging
  - Routine operation further automated each year, even in the last run - as experience directed and as (scarce) developer time was available

Flexibility and maintainability

- Essential ingredients:

  (We did *not* always get these right!)

  – Ability to test new releases or patches with near-zero overhead, making use of any incidental downtime

  – Ability to apply patches without losing provenance of data

  – Both *backward* and *forward* compatibility in data stream

    - Essential to be able to introduce new data, data types into the persisted data *on demand* without waiting for updates to downstream systems

  – Rich electronic logbook that preserves this as a data quality record

  – Continuity of responsibility: professionalization of key software

    - Much subsystem-specific, physicist-provided online code eventually had to be taken over by a core group (for rewriting or just maintenance)
    - Collaboration service can be a false savings

The long haul: beyond construction and commissioning

Some lessons in code management

- Dependency management is the fulcrum of everything
  - Proper abstraction, encapsulation, use of generic programming, separation of software into packages, attention to forward and backward compatibility, code/configuration entanglement
  - When we did this right, the benefit was always measurable
  - Many of our most serious problems came from getting this wrong
  - The ability to decouple changes from each other avoids problem *a priori* but, even more importantly, keeps you nimble
  - If this is under control, you can redesign around performance problems, introduce new features, fix bugs in midstream, etc.

Lessons in code management

- Investment in a highly qualified pool of core people can be difficult to justify on immediate project-specific grounds, but pays off many-fold
  - People who can do intensive debugging, code reviews, understand details of performance measurements are **gold**
- Pushbutton testing frameworks are invaluable; time spent building them pays off on < 1 year time scales
  - Confidence in deploying new releases quickly is invaluable

Thoughts on performance

- Sustained efforts chasing 0.5% performance effects are good investments
  - Even in 2008, 4 FTE-months of talented effort of this nature saved us O(20%) - thousands of core-months of CPU and scaling headaches
- A large, budget-limited system will always be operating near its scaling limits
  - Small improvements have amplified payoffs in throughput
  - Trading off some peak performance for reliability is often worth it
- But nothing beats designing for performance from the start
  - The opacity of C++ performance for neophytes made this difficult
  - The lack of high-quality instrumentation until ~2005 made this worse

Dependency management and external software

- Frequently we confront the question: reuse or buy external software, or redevelop it in-house?
  - Sometimes, the upfront costs seem to favor reuse/purchase
- Managing computing for a 15-25 year initiative is not a one-shot decision - it's an "iterated game", in which the best strategy may be different
- We have learned repeatedly, though hard experience, the virtues of "being in control of our own destiny"
  - *Every single piece* of external software that ended up being coupled to *BaBar* software (at compile or link time) has reduced our nimbleness, restricted our choice of platforms, tangled code development, and resulted in painful migrations

What is the lesson?

- We divested ourselves of several such dependencies, e.g.,
  - Objectivity
  - Rogue Wave Tools.h++

  And we never looked back...

- There is great value in the import of tools, when they provide functionality worth many FTE-years of effort to duplicate
  - We could not do without GEANT4, nor, at least after the phaseout of Objectivity, without ROOT...
  - But they come with a **cost**, and it is not small, and **it grows with time**, in inverse proportion to the availability of funding for in-house FTEs to deal with the resulting problems late in the life of a project.

- Think this through in advance!

Providing resources:
coordinating international contribution

Coordinating international resources

- No regrets: *BaBar* did this really well and it paid off
- Reality: funding agencies would/could not funnel all of the $10Ms needed for computing hardware and professionals to the host lab
  - Without this help we could not have gotten the physics done! It was a truly international effort.
- Mechanisms were devised for the collaboration to document its needs, and incentives for the non-host funding agencies to provide resources at their home labs
- *BaBar* then learned how to make distributed computing work

BaBar's Tier-A and Tier-C resources

- BaBar computing was divided among a set of "Tier-A" sites:

  SLAC, CC-IN2P3 (France), INFN Padova & CNAF (Italy),
  GridKa (Germany), RAL (UK; through 2007)

  - Responsibility for core computing (CPU & disk) provision divided as ~2/3 SLAC, ~1/3 non-US
  - Tier-A countries delivered their shares largely on an in-kind basis at their Tier-A sites, recognized at 50% of nominal value
  - BaBar operating and online computing costs split ~50-50%
  - Planning to continue replacement, maintenance through 2010
  - Simulation also provided by 10-20 other sites, mostly universities
- Analysis computing assigned to Tier-A sites by topical group
  - Skimmed data & MC is distributed to sites as needed for this
- Specific production tasks assigned to some sites as well
- Roughly half of BaBar computing was off-SLAC 2004-2007

Getting the physics done: working with analysts

Working with analysts

- Success is a powerful argument...
  yet I think one could really do better in this area
  - Problem: hard-cash investments in FTEs are needed in order to produce difficult-to-measure improvements in physicist productivity
  - Physics data interface in C++ was much, much more difficult for people to learn than the equivalent Fortran + HEP hacks of the past
    - That was plainly not a problem inherent to C++
    - Solution: plan & budget for clean-sheet redesign at some early stage
  - Data catalog took years to became even reasonably well aligned with analyst needs
    - Requirements were known but quantity and quality of FTEs not available
  - Documentation had the usual problems: stale, wrong, and off-point
    - Intro-level (“Workbook”) was good, most of the rest disappointing
    - Wiki approach would probably have helped a lot

The very long haul:
data preservation and wider access

BaBar after the end of data-taking

- Intense analysis will continue through 2009 and into 2010
- There will be a long tail after that
  - Initially it will look a lot like things do now, with fewer people
  - Eventually a new model is needed to maintain access to data, MC
- *BaBar* as a whole is thinking through the requirements

- The computing group has been thinking about technicalities
  - First step: port to the latest operating systems, compilers, libraries; this will buy us a few years of time
  - Options for very long term access are being explored (data simplification, virtualization of the *BaBar* environment, outsourcing)

Data Curation

- *BaBar* has a unique data sample
  - Very competitive Y(4S) dataset with an excellent detector and tools
  - By far the largest Y(2S) and Y(3S) datasets in the world
  - Very large and clean sample of tau lepton decays
- Many new-physics signals could be visible in this data
  - The mass scales being investigated at the energy frontier are accessible in many channels at BaBar, both at tree and penguin levels, producing enhancements in rare decays and/or distortions in the pattern of CP violation and the unitarity triangle parameters
  - We are already looking for those that seem most promising *a priori*

However...

- Discoveries at the Tevatron or LHC (or beyond) could point in new directions, even many years from now

Planning for the very long term

- Many of us in HEP are concerned about maintaining very long term access to legacy data sets
  - Goal: to be able to revisit old data in the light of future discoveries
  - This seems to me to rise to the level of a moral obligation to our peers and the fellow citizens who funded us
  - At the ICFA seminar this week, tomorrow at 17:15, there will be a Round Table on

    "Relaxing the proprietary ownership protocols for HEP data"
    - I will be speaking there, on "Preserving and Expanding Access to Legacy HEP Data Sets", and will discuss a new initiative...

Workshop series on "Data Preservation and Long Term Analysis in HEP"

- Representation so far from H1/ZEUS/CDF/D0/*BaBar*/Belle and computing centers from DESY/SLAC/KEK/FNAL
- First workshop to be held at DESY in late January

Final thoughts

- BaBar pioneered many aspects of the modern large-scale computing strategies for HEP experiments
- Despite the risks of being an early adopter, we accomplished everything we planned, and more, with the B-Factory data
- We could not have done it without *very* generous support from our funding agencies
- Come back this afternoon to see the fruits of our labor!