CDF Run2 offline and computing

(10 years in 10 minutes)

Stefano Belforte
INFN – Trieste CDF

Subtitle variations:

How to be wrong by a factor 100 and still make it

A collection of good old common sense

A success story: no analysis work in CDF has been hampered by lack of computing power or bad software (so far)

No magic recepy given

Why does it apply to you

- Similar complexity to LHC experiment
- Similar data volume (given the time difference)

- 100 Hz (and growing) of data to tape
- 0.5 Pbyte of data/year
- 200TB of data on disk for analysis at FNAL now
- 9 raw data streams feeding ~50 primary data streams
- 100's data sets

- A working analysis grid
  - not all LCG promises fulfilled
  - better then current LCG on some side, worse on others

The numbers (rounded)

- 800 CDF collaborators
- 700 "that run jobs" (users that asked for a queue on analysis farm)
- 10 years of code development
- 6 years of data taking
- 10 years of data analysis
- 1M lines of offline code
- 1TB data logged on tape every day
- 500TB data to analyze every year
- 10^9 events per year to analyze
- 1M files to handle every year
- 1000 CPU's in the analysis farm
- 200 TB disk space for analysis
- 10 remote farms for analysis and MC
- 10K jobs waiting to be executed on a typical day
- Uncounted/uncountable local clusters (50 institutions)
- 100 "librarians"
- 760 "CVS authors" (people who could write in CVS at one time or another)
- 2 years typical lifetime of offline heads

An exercise in chaos management

Is your experiment much bigger ?

"A complex system that works is found to have evolved from a simple system that worked… A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a working simple system"

- G.Booch OO Analysis & Design 2nd ed. Pag. 13

Maybe CDF is such a simple system ?

Foreword

- I found it boring to do the usual collection of numbers and "just describe CDF"
- It is all on the web anyhow
- Will try something new, at cost of giving no information

- **I will give my resummation of having lived Run2 as an offline user, reviewer, developer, planner (but ~never in charge of, I enjoyed "freedom to complain"). And only say a small part of "CDF", I'll be happy to take questions offline.**

- Even if eventually I was one of CDF Analysis farm builders, head of CDF computing in Italy and carrier of various titles/responsibilities in CDF computing, during Run2 commissioning I was actually commissioning a piece of the trigger I had been building the previous 6 years. So I beg you to:
  - Forgive my uncomplete understanding
  - Take this as a complement-to and not a replacement-for other talks you listened to, read etc.
  - Do not ask me any C++ question
  - Indulge me on taking some freedom of expression, and do not tell my collegues what I will tell you today

Disclaimer

- There is no CDF official report of what went well or bad
- We are of course successful (papers appearing on Phys.Rev.), but…
  - since learning stems from mistakes
  - and only new mistakes are allowed for LHC experiments
- …I am here to review some of our worst mistakes ☺
  - Hopefully also a few good examples
  - What to call a mistake is sometimes a matter of opinion
- This talk results from polling a handful of present and past CDF offline and computing heads and a few knowledgeable people at large but things reported today are to be taken as Stefano's personal, biased, incomplete, possibly wrong recollection and wrap-up
  - They do not represent CDF, FNAL, DOE, INFN etc. etc.

History: we had a good plan (circa 1997)

- At the time we were supposed to have 2fb-1 by 2002
- 3 guidelines for CDF computing upgrade Run1→Run2
  - All new code, all new hardware

1. Build on Run1 success
   - Data was analyzed
   - No major drawback emerged
2. Smooth introduction of C++
   - Allow wrapped Fortran and "banks" to survive for a while
3. Fix most acute problems
   - Data access
     - hand mounted tapes
     - scripts with lists of file names
   - Bookkeeping
     - reproducibility of past results
     - offline version + calibration constants

2001: It did not work out

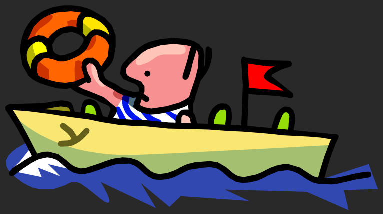
- OO proved much less friendly then previous Fortran
  - Slow learning curve
  - Code harder to read
  - Banks→objects: Documentation from poor to none
    - the hacker's motto "use the force, read the source"
  - Code quality did not improve
- CPU needs for simple tasks increased x10
- The "Run1 model" for analysis hardware architecture broke
  - Had to change from a few big SMP's to 1000 PC's
- But other parts of offline upgrade did very well:
  - event reconstruction (production farm) OK
    - about as big now (~x2) as we spec'd ten years ago !!!
  - Data Handling OK (heavily revised, but little extra $)
  - bookkeeping OK (for production)

But the plan was fulfilled ($$ spent) on schedule

- 10M$ plan in 1997
  - Run2 was due to end 2002
  - Scale from Run1
  - Big SMP's
  - Fiber Channel SAN
- DONE! All money spent by 2001

- OO came in: CPU/job → x10
- Blamed on persistent→transient objects conversion
- 2001 review: not enough CPU
- New code ? Nope !

- 5 years after original plan:
  - Moore's law to the rescue
  - Linux farms + IDE disks
  - CPU/$ → x 10
- New computing plan 2002-8

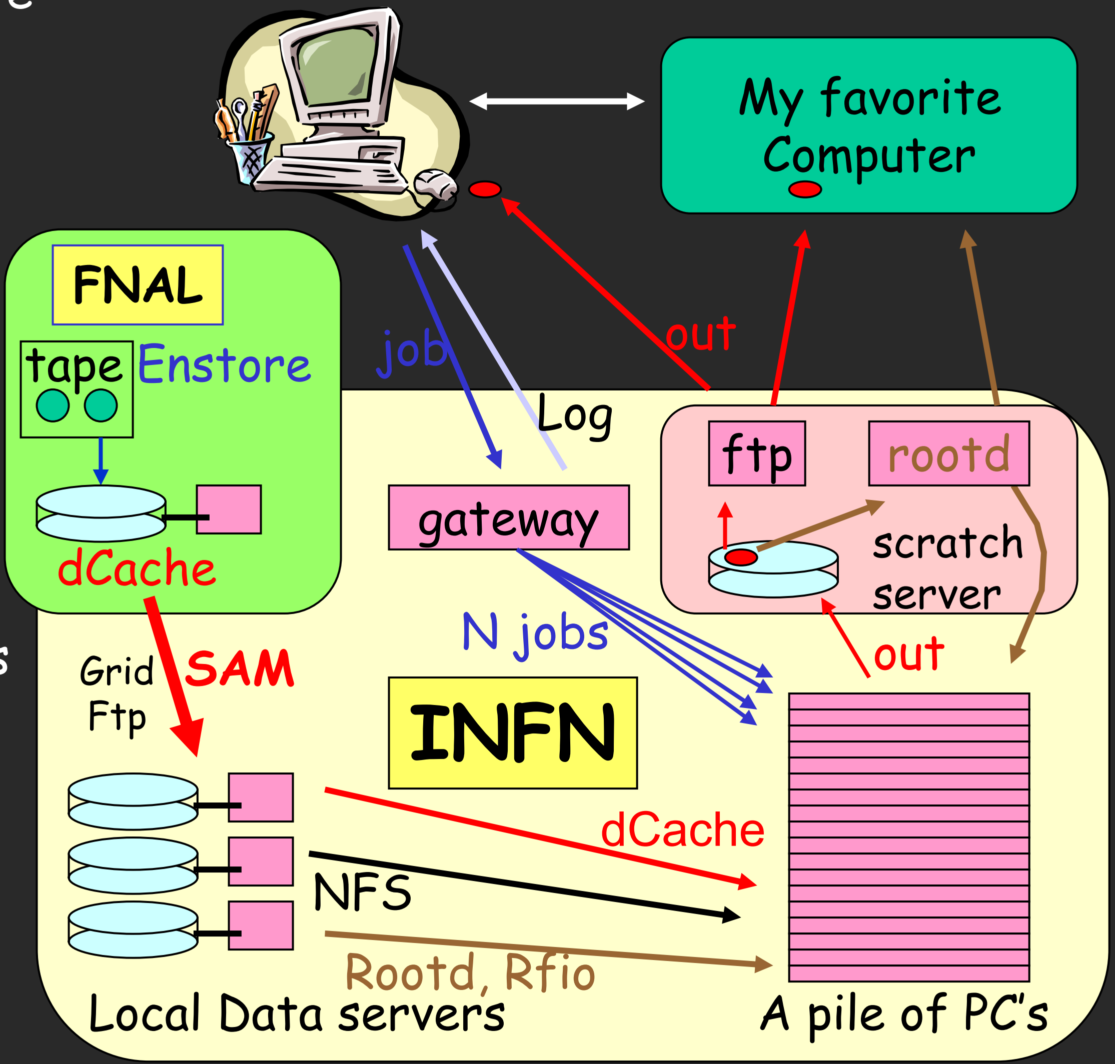


Planning history

- **1997**: needs estimate based on extrapolation from Run1 (big SMP's)
  - Killed by OO and SVT (10M$, fully spent by 2001)
- **2001**: 1 author – 14 pages
  - Needs assessment based on high Pt datasets
  - Part of review of old model (killed by that review) no cost estimate
  - O(1000) CPU + O(100)TB to analyze 2fb^-1
- **2002**: 10 authors– 27 pages
  - MC still an unknown, based on "data scale with L" (only true for high Pt)
  - request 2M$/anno a Fnal
- **2003**: 24+ authors– 67 pages
  - request 3M$/anno a Fnal
- **Nowdays**: still recycling 2003 model for needs estimate, even if we know since last year that it is wrong. Party line is "give us all you can, we'll use all CPU and disk we can lay hands on". (all CDF analysis farms 100% used at all times)

The CDF Analysis Farm (aka CAF): http://cdfcaf.fnal.gov

- Compile/link/debug everywhere
- Submit from everywhere
- Execute on the CAF
  - Submission of N parallel jobs with single command
  - Access local data from CAF disks
  - Access tape data via transparent cache
- Get job output everywhere
- Store small output on local scratch area for later analysis from everywhere
- Great monitor (see Web site)
- USERS LOVE IT !!
- 2 CAF's at FNAL (1000 CPU's)
- CAF's in Italy, Japan, Taiwan, Korea, SanDiego, Rutgers, MIT, Canada, Spain… almost double FNAL offer



Why is the Cdf Analysis Farm so big ?

- In the end ~100x the CPU spec'd in 1997
- Code for "event dump" 10x slower
- Still with 1/10-th the foresought integrated luminosity of Run2 we had bought 100x the planned computing power, 10x the disk and have turned to "GRID" to get more of both

- What else had gone wrong ?
  - Data is not proportional to luminosity
  - Run2 is not simply 20x Run1
    - 1st Run2 physics papers are about charm physics
  - We did not have hadronic B's in Run1, and did not know (and are still learning) how to deal with that huge data sample
  - Most user analysis code uses as much CPU as full reconstruction
  - Users run more, larger jobs then in Run1
    - we extrapolated from "mature" Run1, most Run1 data came after many years of refinement in analysis procedures

Do we waste CPU ?

- Once we started giving CPU to the masses, they took it in hunger and we had no capability to control and guide behavior.
  - Since I can run my code on "the sample" in a week I will not bother to optimize, test on small subsamples, share, document…
- Buying hardware has proven easier then imposing discipline on physicists
- Is that really bad ? Is freedom of invention a bug or a feature ?
- Do people really waste resources out of carelessness ?
- User Joe runs on "the sample" 8 times to get the result, would 4 have been enough ?

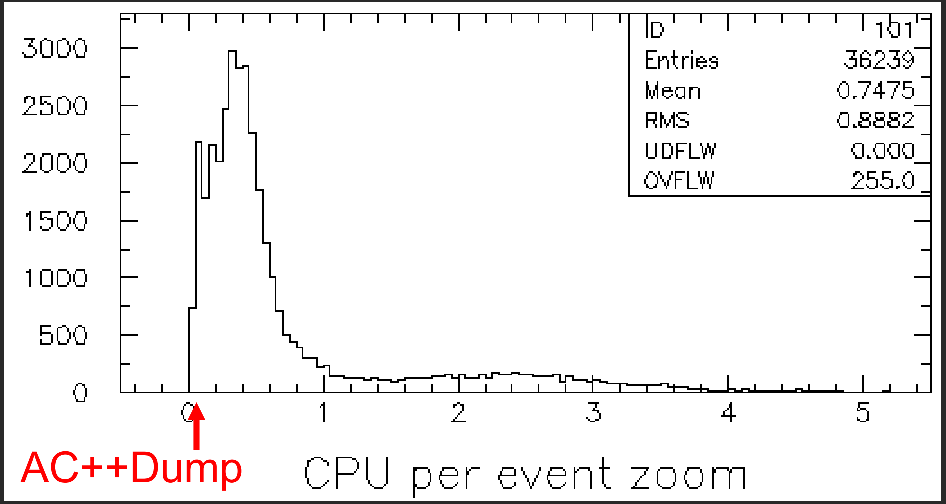
- CDF conducted review of "efficiency and optimization of resource usage" in summer 2004. (Under funding agency pressure)

Computing Usage assessment (2004)

- Much more user MC done/needed then planned
  - QCD/EWK alredy doing almost all computing on ntuple
  - Top/Exotics do similarly, but also do a lot of MC and run a lot on data to develop good algorithms (e.g. jet b-tag)
  - B analysis are enourmously CPU consuming due to combinatorial load in secondary/tertiary vertex finding (4-track vertexes e.g.)
- CPU usage almost evenly spread in hundreds of users, everybody's work would need to be optimised to get an impact
- Most user's jobs using much more CPU/event then expected
- Ample anedoctal evidence for throw-away work due to stupid mistakes, poor documentation etc., but no quantification possible
- Many users doing many different things. No way to isolate typical cases and attack them. Very difficult to optimize.
- Users are using CPU to try, test, explore, learn, do physics

Browsing 8 month of CAF log files

- Look at CPU seconds spent on each event, compare to bare "event dump"



AC++Dump sits at ~0.06 sec/event

- 40% of total CPU from jobs that use >1 sec/event (more then full reconstruction): have ideas on how to help here but possible saving only <10%
- 60% of total CPU from jobs that use <1sec/event, each job minuscule fraction of total
  - Not I/O limited, user code is the "culprit"
  - could not find a way to summarize/break-down them

Suggested path to efficiency: herding cats

- Move more work from scattered users effort to planned effort
  - Give a lot of resources to planned jobs
  - Squeeze random user work to force optimization by necessity:
    - make waste expensive for user
  - Since planning requires effort over and beyond writing a thesis, not clear where the needed manpower may come from
- Move "tasks" from user's executable to production (cosmic ray, dedx, beam constraint): need validation, see above comment

- Need time and effort from physicists not computing professionals
- Money can only help by buying computers

- How to build an environment that promotes this ?
  - Do not take CDF as an example !

Pontification time

- Now a survey of wisdom from my heads and peers
  - Anonymous of course

“Trivial” mistakes (that happened nevertheless)

- Get locked in old technology
  - Hardware, software ..
  - You will need to change, and again

- Ignore deep pockets
  - Beware of your simple solution that keeps out of large, difficult to work with, organized groups (your favorite lab computing department e.g.)
  - Needs will change (grow)
  - Porting to new hardware/software will be needed
  - People who now “can do it all by themselves” will leave

- CDF completely turned around on storage, data handling, analyis hardware and software in a 2~3 years time span

Do not overshoot

- Do not look for the perfect solution to a problem that does not need one, good enough is enough
- What we do is simple, we only have to do it on many events many times
- Example:
  - Event data model unnecessarely complex
  - Bad things follows:
    - Maybe users code is so slow because they do not even dream of understanding what they are doing and take the hit rather then try to fix it ?
    - Maybe we could have spent less time on cool OO stuff and more on giving users good tools, examples, habits, high level data and methods ?

Do not panic (when you see data)

- Do not rush a solution from the top, hiding troubles under the carpet at the first sight of data
  - E.g. do no rescale MC, go the hard long way and understand it
- First impact with data (and users) will shatter all planning, but still have some time before you produce real physics
- Examples:
  - 5 years after we sacked data handling manger because (among other things) it was difficult to get at data on tape, and changed the Data Managemnet tools, users still need to apply by mail before they can analyse a tape resident dataset
  - Under "conference pressure" we bought all disk (~300TB) we could get, and now have no idea how much most of the data there are used, when it is the last time it was used, and sometimes even simply what data is there

Some of the worst things we did

- Attempted to develop a data handling system independently of D0. This unncessarily sapped resources from both CDF and D0
- Failed to provide users with simple tools to manage the analysis of large datasets (ie, highly parallelized analysis jobs). This deficiency has not only reduced the efficiency of all the people working on analysis, but also caused major problems for offline operations
- Failed to provide a strong set of easily available debugging and code analysis tools. This has cost countless hours of people's time
- Failed to provide users (physics groups) easy ways to attach custom data to the event, since we do not save candidates, heavy combinatorial selections are repeated over and over as event samples get reprocessed (even after being skimmed)

More of the worst things we did

- Wrote many data objects that are far too complicated. The lowest level of data structures should be very simple with no unnecessary features or levels of abstraction. Algorithm classes should be written to calculate more useful quantities if needed. Tracks should not know how to fit themselves or be part of an interface for vertex fitting! Electrons need not know how to find the event vertex and correct their own Et!

- Developed an event data model that had the event persistency mechanism inseparably built into it. This is actually an example of introducing unnecessary features. Had the objects of the event model not been tied to the persistency mechanism, and had the objects themselves been fairly simple, then the entire reconstruction could have been trivially ported to other contexts: the same structures used in AC++ jobs could have been used in ntuple-based analyses; the reconstruction code itself could have been run in an ntuple-based analysis without any modification; and etc.

Some of the best things we did

- The CAF and its associated submission and monitoring tools
- Moved away from simple banks to more structured data types for event data
- (Eventually) Adopted data handling tools supported by others (Fermilab)
- Created lots of sensibly defined production output streams
- Most importantly, wrote good, reasonably fast reconstruction algorithms in time to run them all in production.

Parting words

- Need to be intelligent and walk the fine line
  - Just make it complicated enough to be useful without being obnoxious
  - Never marry a solution, and always keep it simple

- Every feature has a cost, if not money, human time
  - Will the phyisics really benefit ?
  - Will time to publication shorten ?
  - Will some uncertainety decrease ?
  - Will a new measurement be possible ?

- ~~OBJECT~~ GOAL ORIENTED SOFTWARE
  - GOAL is Phyiscs

Further readings

- I have to confess I did not find the time to do the research and write this
- Of course you can just look it up on google
- Anyhow I promise a page of URL's later on when I get online, to be added to the meeting agenda