Some Questions and Answers Related to the Performance Comparison between ROOT, Objectivity/DB and HistOOgrams

---

On this page we try to give some answers to issues raised concerning our recently published "Comparison between ROOT, Objectivity/DB and LHC++ HistOOgrams" paper. This page is in a question/answer format (some questions are more like statements, however).

Q: Why did you use a 2 KB page size for the Objy database?
A: We got the populated database exactly as we reported from Dirk Duellmann of RD45. Why he used 2 KB? We don't know. Maybe the file would have been twice as large when using 8 KB? We hope you don't have to use one page size for a size benchmark and another for a performance benchmark.

Q: Why did you use a 32 KB page size for ROOT while the Objy page size was only 2 KB?
A: The ROOT TTree 32 KB size is the size of the in memory buffers used by the TTree during filling. When a buffer is full it is compressed and written to disk. You probably can compare this with the buffer memory used by Objy before a commit. We don't have the concept of pages in the ROOT database system. Previous experience has shown that using a fixed page or block size has many limitations. It is often either too big or too small and hardly ever optimal (see ZEBRA/RZ). By the way, a 32 KB buffer containing identical constants will result in a single word on disk.

Q: Why did you not access the Objy database via the well tuned AMS server?
A: Both the ROOT and Objy database were on the same NFS partition, no advantage for either one. We did not use AMS because there were no AMS servers set up at CERN for ATLAS, at the time of the benchmark. If the performance is much better via AMS, great. Maybe in that case we can compare Objy with PROOF (Parallel ROOT Facility), with which we can traverse a large set of ROOT databases in parallel. This process is transparent for the user and works pretty good in the NA49 environment. Especially on a cheap Linux PC cluster.

Q: ROOT databases do not scale.
A: Where this idea comes from we don't know. For more than a year we use in NA49 routinely ROOT databases of more than 70 GB (with single files up to 1.5 - 2 GB) to make final analysis. The system behaves perfectly well under these circumstances. There are no limiting parameters in the ROOT DB system that could prevent scaling to the expected LHC database sizes. Currently one single database can

not be more than 2 GB. We plan to support 64 bit file systems in the coming weeks. We don't know yet of a single experiment that uses currently Objy on this scale for data analysis.

Q: You should only have used tags when doing the queries.
A: Tags are an artifact introduced by the RD45 team to circumvent the poor performance of Objy when reading selected parts of an object. Tags break completely the OO model and brings you straight back to the PAW CWN model where you work on a simple table of attributes. What have you gained in that case? You only introduced C++ as syntactic sugar without profiting from the full OO principles of data hiding, encapsulation, inheritance, etc. Here follows a simple example. Imagine you have the following class Event:

```
class Event {
private:
   int fRun;
   int fNpart;
   ...
   ...
public:
   int   GetNparts() const { return fNpart; }
   float GetThrust() const { /* complicated calculation */ }
   float GetOblateness() const { /* complicated calculation */ }
   ...
};
```

Now what you want to be able to do during your analysis is, e.g.:

```
for (all events) {
   GetEvent(i);   // Get event pointed to by evt in memory
   ...
   if (evt->GetNparts() < partCut)
      thrusth.Fill(evt->GetThrust());   // Histogram thrust for selected events
   ...
}
```

How would you do this when all event attributes are independent tag variables? The point is that the concept of an object has been completely lost and that the internals have been fully exposed via the tags.

Next, assuming you accept the usage of tags, what becomes a tag and what not. The beauty of ad-hoc data mining and analysis is that often you don't know what you are looking for. You try a lot of different things. Do you want to make for each physics team or each user a special version of a tag database? A tag database might be useful when it is used to reference objects residing in large databases that migrate to mass storage devices. However, in that case the tag DB should try to avoid duplication of the original data, and rather contain a compressed summary of the event information. ROOT supports this via an event list mechanism that can be used for looping, in a very efficient way, on a subset of the original database.

Q: Why did you only put 9 variables in the tag DB? No wonder the results are bad for Objy.
A: The classes were given to us by RD45. We modeled the Event and Particle classes for the ROOT version after this Objy example. As we said in the paper we did not use tags because the ROOT performance without this trick is good enough. However, if we make every attribute a tag (as is automatically done by h2root) the ROOT database traversal timings, as reported in table 1, are 50% shorter.

In the case a full event is read from the ROOT DB and not only a few selected attributes the difference with Objy is less lopsided. ROOT takes in that case 100 sec where Objy takes 122. Not anymore a factor 5 but still 18% faster. The point, however, is that the ROOT DB is flexible enough to support both modes of working (whole event vs. subset) using a single database.

Q: Is there not a more "PAW like" way to obtain the three histograms as produced in the benchmark program?
A: Sure. Using the interactive root executable module one can obtain the same three histograms by typing the following commands:

```
$ root
root [0] gSystem.Load("libAtlasProd.so")  // load application specific libs
root [1] TFile f("event.root")            // open event db
root [2] evt.Draw("Npart")                // histogram Npart and draw histogram
root [3] evt.Draw("particles.mass")       // histogram mass of each particle
root [4] evt.Draw("event.GetTotMass()")   // histogram total mass of event
root [5] evt.Draw("particles.mass", "Npart<500") // now we start playing...
```

These same commands can also easily be executed as macro in batch. We don't know what the LHC++ equivalent is.

Q: Why did you not verify your results with the LHC++/RD45 team?
A: We did. Because of the unbelievable low performance we verified with a member of the LHC++ team the compile options of the histOOgram library. He confirmed that the released histOOgram library in the AFS pro directory was compiled in optimized mode. Anyway, if it was a development version you would gain maybe 30 - 50% by optimized compiling but not a factor 40! Further we discussed all results with this person and he confirmed that our results matched with what he expected based on his own tests.

Q: You compared the size of a compressed ROOT DB versus an uncompressed Objy DB.
A: Is a product allowed to use all its features? ROOT compresses by default and it is an integrated feature of the system. The compression is not done a-posteriori. We think it is total nonsense that a system is not allowed to use its basic features, because they are not supported by the competition. That is just why one product is better than another.

Q: Why did you not tell the LHC++/RD45 team you were going to make a comparison?
A: We did what everybody could have done. V1.0 of LHC++ was publicly released Nov. 4 (the date of the first road show to the ALICE collaboration) and everybody was invited to try the programs. That is just what we did. To be not unfair we only used a database provided by the LHC++ team and not one that we produced ourselves. In case the LHC++ team released debug versions of their libraries then that was pretty stupid or naive. Anyway, as mentioned above, we verified the results with somebody from the LHC++ team before publication.

Q: You are comparing the ROOT "production" system versus the first LHC++ prototype.
A: We probably should consider it a compliment that ROOT is labeled a "production" system. We have been designing and building our "production" system during the last three years with effectively a total of about 8 man years. The current ROOT team is only 2 people. We are supporting more than a 1000 users, provide versions for all Unix/Linux platforms with at least two different compilers and for WinNT/95. Further we are actively working on new developments, we give training courses and also help experiments getting started with C++ (e.g. ATLFAST++). The LHC++/RD45 team is substantially

larger and has also been working for the last three years on their product. The LHC++ "prototype" was taken from a directory named 1.0 (and not 0.1 or proto).

Q: ROOT has no support structure. With commercial software comes professional support.
A: Anybody ever thought about user support in the LHC++ framework? Imagine your program crashes. You spend some time analyzing the problem and find it crashes in some LHC++ provided library. You send a bug report to the LHC++ team. They verify and forward the bug to the vendor. The vendor responds that it is not reproducible and claims that it must be a problem in some other part of the system. The other vendor says the same thing. Finally somebody acknowledges the problem and says it will be fixed in the next release in 6 months. You cancel the presentation of your paper because you can not finish your analysis in time. Nightmare or reality?

Q: A "home grown" solution is too expensive to be supported by CERN.
A: First of all, compare the manpower costs of both systems after 3 years, 8 man year for ROOT versus "a lot more" for LHC++/RD45 (currently the group is several times larger than the ROOT team). In addition to this add for LHC++ a fair amount of yearly licensing fees, the necessity of having per experiment a local license administrator/coordinator (task not to be neglected for the large LHC collaborations) and the fact that only a limited number of platforms and compilers are supported (excluding the cheap, powerful and popular Linux OS and gcc compiler). Currently LHC++ supports only a fraction of the features already available in ROOT. For example, features not yet available in LHC++ are: automatic documentation generation tools for C++ code, shared memory facilities, client/server capabilities over which full objects can be send, powerful and compilable command and macro language, a fully integrated graphics system, automatic GUI allowing direct object interaction, etc.

Further, what would be the cost if the database vendor would go out of business (Objectivity is fairly small and had a very hard time recently)? Is the source deposited under escrow? What will it cost to migrate peta bytes of data from one vendor’s system to another (remember ODMG defines a source code and not a binary standard)?

Q: A "home grown" solution is not flexible enough.
A: Not having the source means one has to wait till the different vendors release compatible libraries for each new OS version. Also one can not take advantage of new pre-release (beta) hardware or software that is often offered by vendors to CERN and other physics institutes. Having the source means you have full control and you can react early to possible new trends in hardware and software.

Q: Do you have an interface to HPSS?
A: If there is a well defined API available for HPSS then we do not foresee any major problems in integrating HPSS into the ROOT DB system.

Rene Brun, Nenad Buncic & Fons Rademakers
Last update 1/12/97 by FR