INTRODUCTION TO STORAGE AND SOFTWARE SYSTEMS FOR DATA ANALYSIS

Bob Jacobsen
University of California, Berkeley, USA

Abstract
The Storage and Software Systems for Data Analysis track discusses how HEP physics data is taken, processed and analyzed, with emphasis on the problems that data size and CPU needs pose for people trying to do experimental physics. The role of software engineering is discussed in the context of building large, robust systems that must at the same time be accessible to physicists. We include some examples of existing systems for physics analysis, and raise some issues to consider when evaluating them. This lecture is the introduction to those topics.

1. INTRODUCTION

Many modern high-energy physics (HEP) experiments are done by collaborations of hundreds of people. Together, these groups construct and operate complex detectors, recording billions of events and terabytes of data, all toward the goal of "doing physics". In this note, we provide an introduction to how we currently do this, and raise a number of issues to be considered when thinking about the new systems that are now being built.

2. SCALE OF THE EXPERIMENTS

BaBar, CDF and D0 are examples of the large experiments now taking or about to take data in a collider environment. The collaborations that have built these experiments contain 300 to 600 members with varying levels of activity. Almost everybody is considered "familiar" with using computing to do their work, but only a small fraction of the collaboration can be considered as computing professionals. Some of these can even be considered world-class experts in large scale computing. The LHC experiments expect to have collaborations that are a factor of 3 to 6 bigger, with similar distributions of effort.

The scientific method can be summarized as "hypothesis; experiment; conclusion". From proposal to final results of a modern high-energy physics experiment takes several decades. The time-scale of a particular physics measurement is shorter, however. Typically an analysis will take about a year from start to finish, from the idea to measure a quantity to the final result. During this time, the people doing the analysis have to understand their data and the detector's effect on it, connect that to the physical process they want to measure, develop tests of consistency, etc. Much of this work involves the analysis of data from the experiment, and it is here that the computing systems for doing data analysis have direct impact. If the analyst can pose and answer questions quickly, the process of understanding the data and making a measurement will happen faster. If the analysis tools prevent posing sufficiently complex questions, such as detailed examination of the effects of the detector and algorithms, then hard measurements will be made more difficult.

These experiments record large data samples. In the case of BaBar, over a million physics events are recorded each day. All three of the example experiments will be recording hundreds of Terabytes each year for analysis. A large fraction of the collaboration wants to do that analysis, preferably at their home institutes, so it must be possible for a hundred people to simultaneously access the data at locations spread across four continents. Figure 1 shows that even the exponential

increases in cost/performance predicted by Moore's law are not enough to provide the computing power that these growing data samples will need. We'll just have to get smarter!

Modern experiments generally expect to run for almost all of each calendar year. It is no longer possible to process a years worth of data long after it has been taken, allowing time for the alignment and calibration to take place first. Instead, the experiments are moving to "factory" mode of data-taking in which data must be processed and reprocessed on an ongoing production basis.

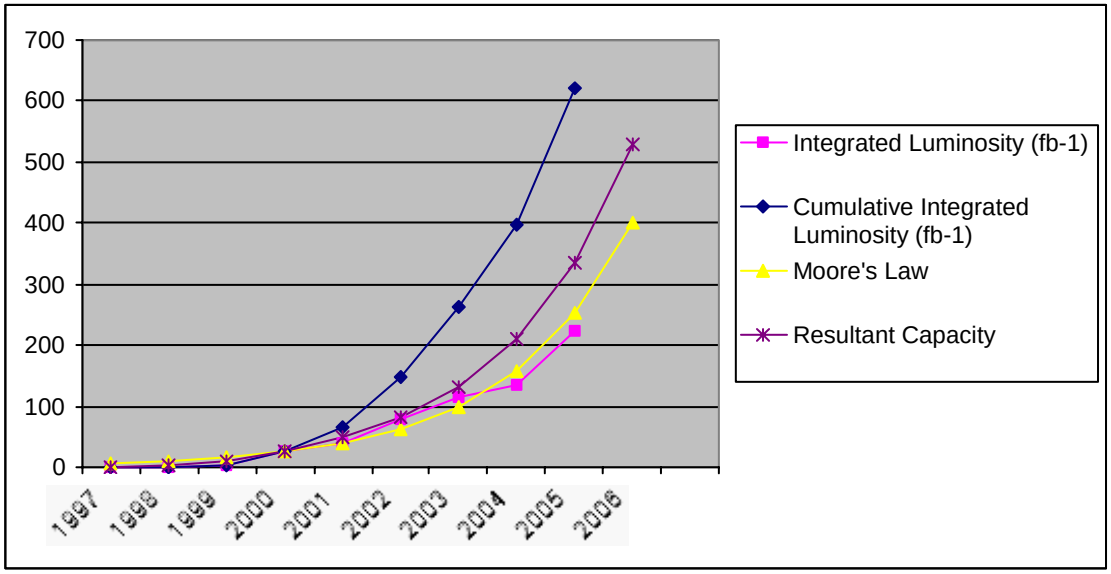

Figure 1 – Moore's law vs. luminosity growth at the BaBar experiment. Figure courtesy R. Mount, SLAC.

3. ANALYSIS MODELS

When large-scale computing was introduced to experimental high-energy physics, operating in "batch mode" was the only available option. This resulted in an analysis model (Figure 2) where the data from the detector was placed in a large tape store, and repeatedly accessed by individual batch jobs. As time went on, we learned how to keep interesting parts of the data on disk storage for faster access, and to use interactive computing systems for developing and testing the programs. This model was used for many years, and produced a lot of important physics results.

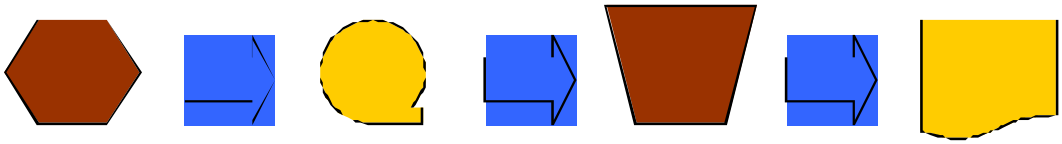

Fig. 2 The batch analysis model. Data moves from the detector (hexagon) to a tape store. Batch jobs are repeatedly run to create the final output of numbers and histograms.

Approximately 15 years ago, the introduction of PAW popularized a new "batch-interactive" model (Figure 3). Large mainframe computers were used in batch mode to efficiently produce summary data in a form that can efficiently be used by smaller workstation computers. This summary data is then interactively processed using a different set of programs to produce the final results.

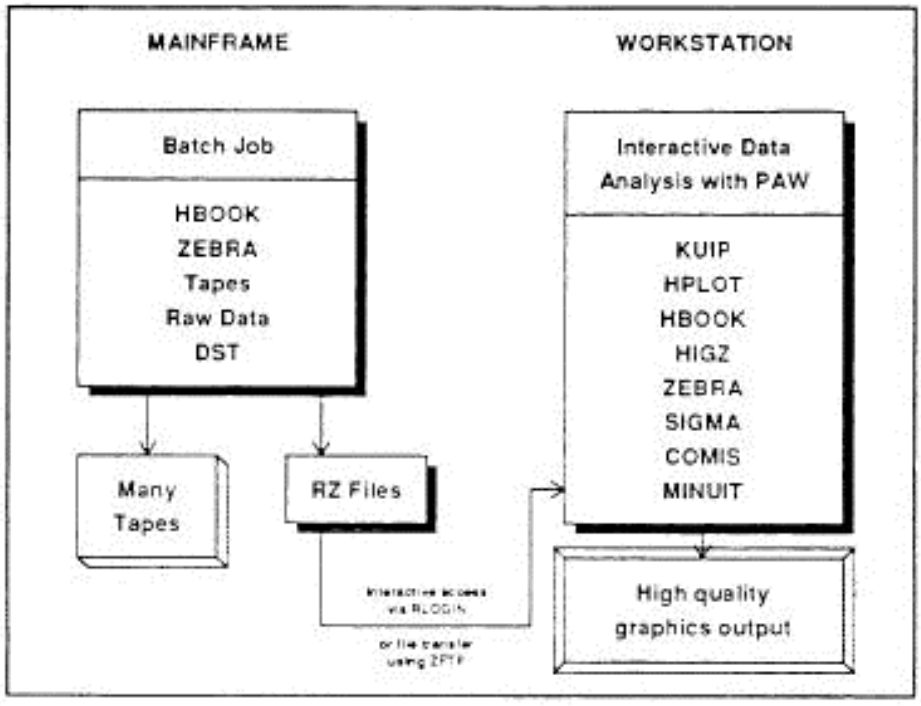

Figure 6.2 Schematic presentation of the various steps in the data analysis chain

Fig. 3 The batch-interactive analysis model. This figure was taken from the PAW manual.

Like many revolutionary advances, this brought both improvements and the seeds of eventual dissatisfaction. The rapid turnaround made possible by interactive workstations made it possible to "PAW through the data", adjusting cuts and improving understanding. This was a qualitative improvement, not just a quantitative one, because the analyst could now make plots to answer many simple questions about as quickly as the plots could be understood. Speeding up the hypothesis-experiment-conclusion loop allowed a more complete understanding of detectors and data than had previously been possible, and allowed more complicated effects to be studied.

Within the past few years, several areas of limitation have been identified with this model. The first of these is the limitations of doing interactive work on individual workstations. We have learned to replace production processing on a single computer with large farms of comparatively cheaper machines. This, combined with the cost/performance benefits described by Moore's law, allowed us to (barely) keep up with the growing size of our data sets, and the increasing complexity of the algorithms we use. Instead, the desktop workstation has become the bottleneck. Second, as more and more complex questions are posed, the need increases to use the same computational tools for interactive analysis as for the large-scale production processing. It is no longer sufficient to look at arrays (ntuples) of numbers with simple cuts; we now want to run large parts of our simulation and reconstruction algorithms as part of our final analysis. This implies that we need to build analysis systems with closer connections between the "batch production" and "interactive analysis" components. In addition, the long time-scale of the LHC experiments has caused concern about the expected lifetime of their software. In the past, typical experiments ran for five or at most ten years. By the end of that time, they had typically outgrown their initial computing systems. Collaborations have historically had great difficulty moving to newer systems while at the same time taking and

analyzing data. The expected twenty year lifetime of the LHC experiments means that the collaborations must have ways to extend and replace their software as needs and technology advance.

As we move toward the larger needs of the LHC experiments, a number of different approaches are being tried to improve the situation; those approaches are what we examined during the Storage and Software Systems for Data Analysis track at the 2000 CERN School of Computing.

4. A SCALING MODEL

The size of a problem affects how best to solve it. For example, teaching one person a particular topic can be done well by having them walk down the hall and ask questions of an expert. This breaks down if sixty people need the knowledge because the expert does not have the time, and typically not the patience, to answer the same questions that many times. A CERN School of Computing involves a lot of logistical and organizational work, but is very well suited to teaching a half-dozen or so topics to about sixty people. For a much larger number, perhaps several thousand, a CERN School is impossible. That many students would want to learn a number of different subjects, at different levels, on different time-scales. Universities have developed to handle that kind of education. We infer from this example that qualitatively different solutions can be needed when the problem size is very different.

We can simplify the issues involved by considering a more abstract case (Figure 4). A particular solution to a problem has a "start-up cost", a region of application where the effort grows approximately linearly with the size of the problem, and a region where the solution starts to break down resulting in much faster than linear growth.

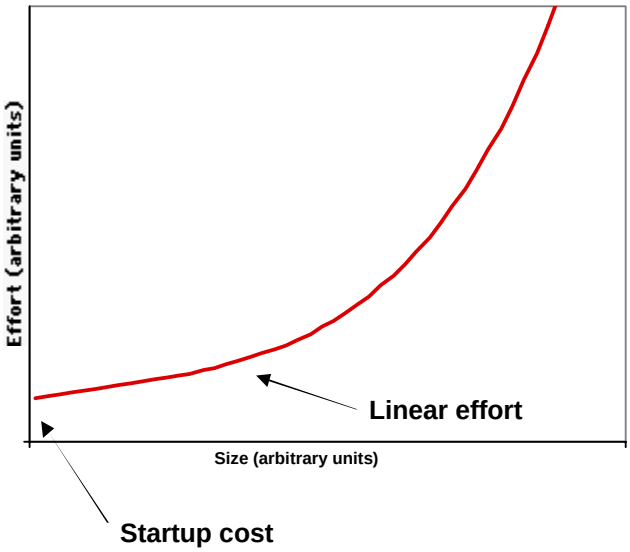

Fig. 4 A model for the cost of solving problems of varying sizes

Now consider what happens when multiple solutions are available for a single problem (Figure 5). For solutions of very small size, clearly the one with the smallest startup cost is the right one. For larger problems, however, you may be better off with a solution that has a flatter curve, i.e. lower slope in the linear region, even if the startup cost is higher.[1] Using a particular solution on problems that are larger than the solution's linear region is almost always a mistake; the solution will start to break down, and the effort needed to keep it functional will grow without bound.

[1] Empirically, we find that almost all solutions that are suited for larger problems have a higher startup cost. Its usually harder to build a system that will scale-up well.

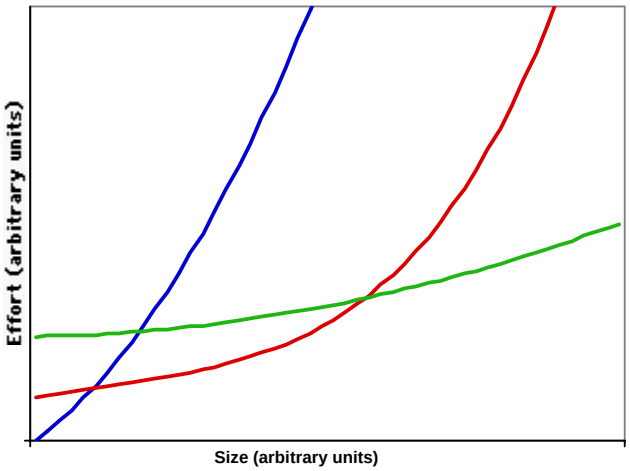

Fig 5 Tradeoffs between different solutions to problems of varying size.

Notice that the size of a problem may change with time. A physicist may write a "quick and dirty" program to solve a specific problem that is not expected to recur. In that case, minimal effort is certainly justified. But that program might become popular, and additional people might want to run it, perhaps on data of slightly different format. This is the range of linear scaling, as people make small modifications to extend the functionality. Eventually, however, these changes can start to conflict; Jane wants to use the program on data without certain characteristics, while John wants to add new functionality that makes heavy use of them. The program becomes more complex, slower, or perhaps less reliable, and eventually can't be extended further. Once that point has been reached, it is necessary to transition to a different solution. This can be costly, and often makes people wish a different solution had been chosen from the beginning.

5. THREE ANALYSIS SYSTEMS

The "Storage and Software Systems for Data Analysis" track at the 2000 CERN School of Computing presented details of two systems currently under development: ROOT and Anaphe/LHC++. In addition, the school also had lectures and exercises on the Java Analysis Studio, JAS. The following sections provide brief summaries of the similarities and differences between these systems. For detail, the reader is referred to the lecture papers later in this report.

One common thread in all of these projects is an effort toward "open development". All three projects make the source-code they develop available to the HEP community. All are interested in getting feedback, especially code changes, from users, and attempt to include them when appropriate. All of the projects restrict write access to their code repository to a team of central developers.

5.1 ROOT

The ROOT project was started in 1995 to provide a PAW replacement in the C++ world. The developers had experience in the creation of CERNLIB, including PAW, and were convinced that the size and lifetime of the LHC experiments required a new system for interactive analysis.

The ROOT developers intend it to be used as a "framework" on which an experimental collaboration will build their offline software.[2] By controlling the link/load process, the event loop, and the class inheritance tree, ROOT provides a large number of powerful capabilities:

[2] It is also possible to use the individual ROOT classes and/or libraries separately. For example, experiments at Fermilab are using the ROOT I/O libraries in offline systems without using the entire ROOT framework. Recent work to clarify the structure of ROOT has simplified using it this way.

A sequential object store

Statistical analysis tools for histogramming, fitting and minimization

A graphical user interface, including geometrical visualization tools

An interactive C++ command line

Dynamic linking and loading

A documentation system

A class browser

Numerical utilities

Inter-process communication tools, shared memory support

Runtime object inspection capabilities

ROOT-specific servers for access to remote files

Unique RTTI capabilities

A large set of container classes

These are closely integrated, which makes the power of the system as a whole much larger than can be understood by examining any specific part. Work is actively proceeding to extend the ROOT system in many additional areas, including access to relational and object databases, connections to Java code, classes to encapsulate GEANT4 simulations, parallel execution of ROOT analysis jobs, and others. There are a large number of people actively building on the ROOT framework at several experiments, resulting in a stream of extensions and improvements being contributed back to the ROOT distribution. In effect, the ROOT developers have demonstrated that they can use the large pool of HEP programming talent to build a composite analysis system for the community.

The CINT C++ interpreter allows use of (almost) the same code for both interactive and compiled execution. Users embed their code in the ROOT system by inserting ROOT-specific cpp macros in the C++ definition and declaration files. The ROOT system then uses CINT and other utilities to create schema information and compiled code to perform ROOT I/O, class browsing, etc.

5.2 Anaphe/LHC++

The Anaphe/LHC++ project set out to provide an updated, object-oriented suite of tools for HEP of similar scope to the CERNLIB FORTRAN libraries, with particular emphasis on the long-term needs of the LHC experiment. The strategy is to provide a flexible, interoperable, customizable set of interfaces, libraries and tools that can be populated with existing (public domain or commercial) implementations where possible, and can have HEP-specific implementations created when necessary. Particular attention is paid to the huge data volume expected at LHC, the distributed computing necessary to process and analyze the data, and the need for long-term evolution and maintenance.

Anaphe/LHC++ has defined and is implementing a number of common components for HEP experimental software:

AIDA – abstract interfaces for common physics analysis tools, e.g. histograms

Visualization environment – using the Qt and OpenInventor de-facto standards

Minimizing and fitting – Gemini and HepFitting packages provide implementations, which are being generalized to an abstract interface. Algorithms from both the NAG commercial packages and the CERNLIB MINUIT implementation are included.

HepODBMS – A HEP-specific interface to a general OO event database, used as an object store. The existing implementation uses the Objectivity commercial product.

Qplotter – HEP-specific visualization classes

HTL – Histogram Template Library implementation

LIZARD – an implementation example of an interactive analysis environment

It is anticipated that these components will be used with others, e.g GEANT4, Open Scientist, PAW, ROOT, COLT and JAS, coming from both the HEP and wider scientific communities. Particular attention has been paid to the inter-connections between these, so as to preserve modularity and flexibility. For example, the use of a clean interface and the commonly-available SWIG package makes it possible for an experiment to use any of a number of scripting languages, including TCL, Python, Perl, etc.

Anaphe/LHC++ was aimed at LHC-scale processing from the start. It has therefore adopted tools to ensure data integrity in a large, distributed environment, at some cost in complexity. An example is the use of an object database, including journaling, transaction safety, and location independent storage, for making analysis objects persistent. A physicist writing his or her own standalone analysis program may not see the need for such capabilities, but when a thousand people are trying to access data while its being processed, they are absolutely necessary. Similarly, the project has emphasized the use of modern software engineering practices such as UML, use cases, CASE tools, etc, to improve the quality and long-term maintainability of the code.

Anaphe/LHC++ should not be considered as a unique base upon which an experiment builds a monolithic software system by creating concrete classes. Rather, by identifying interfaces for components likely to be present in analysis software, Anaphe/LHC++ intends to provide a basic structure that can grow and evolve over the long term, perhaps even as HEP transitions from C++ to Java to TNGT (The Next Great Thing).

5.3 Java Analysis Studio

The strategy of the Java Analysis Studio developers is to leverage the power of Java as much as possible because

It provides many of the facilities needed as standard

Its capabilities are advancing fast

It is easy to learn and well-matched in complexity to physics analysis

It is a mainstream language, so time learning it is well spent

It is a productive language, e.g. no time wasted on core dumps

JAS's focus is primarily on the computational part of the analysis task. As such, it uses defined interfaces, called "DIMs", to attach to an experiment's own method of storing and retrieving data for analysis. JAS is not intended as the basis for creating the production simulation and reconstruction code for an experiment. Rather, JAS interfaces exist or are being defined to attach JAS to other parts of common HEP code, including the GEANT4 simulation package, AIDA for histogramming, WIRED for event display, StdHEP for Monte Carlo simulated events, and similar experiment-specific code. The simple "plugin" architecture is intended to make it convenient to add interfaces by adding C++ code to an existing system. Direct connections to existing C++ code, without creating an explicit interface, is currently a weak point of Java.

Java itself provides many of the desired tools, such as class browsers, object inspection tools, documentation systems, GUI and visualization classes, collection classes, object I/O, inter-process communication, etc. This allows JAS to benefit from the efforts of the world-wide community of

Java tool developers, a much larger group than just HEP programmers. As an example, there are ongoing Java efforts to create GRID-aware tools for distributed computation which can be interfaced for use by JAS. It is expected that large-scale tests of distributed analysis using these tools can be done in the next year.

6. SUMMARY AND CONCLUSIONS

The LHC experiments present us with a dilemma. They will produce large amounts of data, which must be processed, analyzed and understood. Current experiments are now solving problems about order of magnitude smaller, but only by working at the limits of the capability of available people and technology. The several analysis systems now under development promise to improve our capabilities, perhaps even change the way we work. All of these systems have proponents and detractors, strengths and weaknesses. They have taken very different approaches to solving the same basic problems. Over the next years, as the LHC experiments develop and deploy their choices for production and analysis systems, the community needs to profit from the best qualities of each of these systems.

ACKNOWLEDGEMENTS

We wish to thank the organizers of the 2000 CERN School of Computing for a stimulating school. We also thank the developers of ROOT, Anaphe/LHC++ and JAS for both their help with the school and their work to provide to tools to the community.

BIBLIOGRAPHY

The Mythical Man-Month; Fred Brooks; Addison Wesley

Peopleware: Productive Projects and Teams; Tom Demarco and Timothy Lister; Dorset House

The Deadline: A Novel about Project Management; Tom Demarco; Dorset House

The Cathedral and the Bazaar; Eric Raymond; O'Reilly

Designing Object-Oriented C++ Applications Using the Booch Method; Robert Martin; Prentice Hall

Large Scale C++ Software Design; John Lakos; Addison-Wesley

The Computing in High Energy Physics (CHEP) conference series is a good source of information on current activities in this area. The most recent one was this past February in Padova Italy. The conference proceedings have been published, and are on the web. Some of the most relevant papers are listed below.

A108 - The Design, Implementation and Deployment of a Functional Prototype OO Reconstruction Software for CMS. The ORCA Project. D. Stickland et al.

A152 - GAUDI - The Software Architecture and Framework for Building LHCb Data Processing Applications. LHCb Computing Group

A245 - The Physical Design of the CDF Simulation and Reconstruction Software. Elizabeth Sexton-Kennedy et al.

A264 - Rapid Software Development for CLEO III. M. Lohner et al.

A326 - The STAR Offline Framework. V. Fine et al.

C103 - Operational Experience with the BaBar Database - Q. Quarrie et al.

C201 - The CDF RunII Event Data Model. R. Kennedy et al.

C240 - Event Data Storage and Management in STAR. V. Perevoztchov

C367 - The CDF RunII Data Catalog and Data Access Modules. P. Calafiura et al.

E248 - Software Sharing at Fermilab. R. Pordes et al.

F033 - CMT: a software configuration management tool. C. Arnault

F175 - ROOT for Run II. P. Canal et al.

F202 - SoftRelTools rev 2 at Fermilab. J. Amundsen et al