



Fermi National Accelerator Laboratory

FERMILAB-Conf-95/355

Run 2 Analysis Computing for CDF and D0

S. Fuess

*Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510*

November 1995

Proceedings of the *Computing in High Energy Physics 1995 (CHEP '95)*,
Rio de Janeiro, Brazil, September 18-22, 1995

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

RUN 2 ANALYSIS COMPUTING FOR CDF AND DØ

Stuart Fuess

*Fermi National Accelerator Laboratory
Batavia, Illinois 60510, USA*

Two large experiments at the Fermilab Tevatron collider will use upgraded detectors in the next era of running. The associated analysis software is also expected to change, both to account for higher data rates and to embrace new computing paradigms. A discussion is given of the problems facing current and future High Energy Physics (HEP) analysis computing, and several issues explored in detail.

1 Introduction

Run 2 of the Fermilab collider, the first in the Main Injector era, is scheduled to begin in 1999. Both existing large collider experiments, CDF and DØ, plan major detector upgrades, with the central calorimeter in CDF and the calorimeter and muon planes in DØ being the major remaining detector components. Associated analysis software will also need to be newly created. Each experiment now has approximately a million lines of code, so the magnitude of the effort is large.

Both the CDF and DØ collaborations are exploring possible analysis computing models for the future. Internal committees are expected to report to the full collaborations in late 1995. Since final decisions have not yet been made, this paper cannot report on the chosen implementations. However, it is useful to comment on the exploration of problems within analysis computing, and to study which factors are of critical importance.

The analysis computing problems faced in the past and expected in the future for these two experiments are similar, and likely mirror the situation throughout HEP. Each experiment now has approximately a million lines of code, and nearly a hundred Terabytes of combined raw, reconstructed, streamed, and summary data. The experiments expect to each write over 100 Terabytes of raw data per year, or more than an order of magnitude beyond existing data sets. While other experiments at RHIC and the b-factories may soon face similar data sets, and LHC experiments most certainly will have even larger sets, CDF and DØ can provide a perspective *now* of operations with multi-Terabyte accumulations. Both experiments operate with over 500-person collaborations, hence have ample experience in the sociological aspects of large collaborations.

In exploring the problems CDF and DØ have faced in the past, perhaps a better understanding can be had of the process of searching for the future methodology of choice. In this paper two issues in particular will be addressed: the need to plan for change and the importance of the link identifier in allowing flexibility.

2 Problems in Analysis Computing

2.1 *Perceived problems*

Let us begin the exploration of analysis computing by examining problems as reported by the typical user. The most noticed attribute is the complexity of the current analysis environment. Contributing to the complexity are the difficulties arising from the distributed development environment present in a large collaboration. As a result of imperfect controls of the development environment, users are often found unaware of changes made by others; as a result of complex interrelationships within the code, these changes can produce unexpected consequences. The combination of a difficult development environment with highly intertwined code results in a painfully slow development cycle.

A consequence of large unwieldy data sets is the introduction of new forms for reduced sets, either by selection of information about each event (a summary) or by selection of particular events (a stream). To meet the requirements of diverse users, there are typically multiple streams and summaries produced. These complex streams divide and often replicate the data sets, such that individual events may appear multiple times. Because these new data sets compete for scarce disk resources, some of them must be relegated to storage on serial media. Finally, a complicated cataloging system is necessary to maintain order. All factors contribute to a user view of the data which may not be simple, and user access to the data which may involve slow and possibly unreliable serial media.

2.2 *Root problems*

After a cursory examination of the preceding problems, one might be tempted to blame any failures on unsophisticated, or even unskilled, programmers. However, one further observation shows that skill levels may be adequate, but other factors drive a physicist's programming styles. Most important, a physicist's activity is often driven by an urgent need to meet a particular goal. To succeed most rapidly, an 'end run' solution is often attempted; that is, in order to quickly converge, it is often most expedient to develop software with a particular limited goal in mind. When all such efforts are collected then convergence becomes difficult. Clearly, the development process was not adequately aimed at the correct initial goals.

Analysis of data organization difficulties seems simpler. Most steps in physics analyses only need small portions of the data. But attempts to legislate reduced or selected data sets do not satisfy all users. In particular, if certain data sets get relegated to serial media then access becomes practically impossible. The root problem might then be determined as the inability of current data management schemes to flexibly and dynamically make available those portions of the data under active study.

3 Problems in HEP Computing

The preceding exposition of general problems encountered in HEP analysis computing only hinted at some of the underlying explanations of why these problems

surface. It is perhaps more instructive to examine the particular features of HEP computing which set it apart from more typical computing tasks.

HEP is unusual in that the customers of any software development tend to also be the developers. This arrangement leads to potential problems. In particular, since a developer tends to be aimed at a direct goal, usually his own physics analysis, the motivation for producing sharable code is reduced. This is a contributing factor to the prevalence of 'end run' solutions, which we've seen as contributing to the complexity of the entire system. Additionally, the often short 'lifetime' of a student or physicist on a particular project also implies that there is a high turnover in the developer base.

HEP computing also appears to be different in that there are distinct problems being addressed at different phases in the process of analysis. It is desirable to share code among the applications developed for each phase, so they need to be considered collectively. The four phases of HEP computing are the software DAQ trigger, reconstruction, data reduction, and physics analysis. In the earlier phases performance is critical, with the necessity for a well controlled and documented code release mechanism. In these steps the operations typically involve information from the entire event. The latter steps operate on combined large data sets, hence data access is of critical importance. During these phases only selected pieces of the event data may be necessary. There is also an overall progression from tightly managed code development to flexible user-oriented activities.

All of the preceding can be encompassed by the realization that a fundamental property of HEP computing is that it must constantly adapt to change. There will be changes in the customer and developer base and their goals. There will be changes in the specific tasks to be accomplished. There will be changing needs over the life cycle of an experiment. There are several consequences of the efforts to deal with changing requirements. Analysis code is certainly more complex to account for the evolutionary aspects of the problem. Multiple versions of analyzed data appear as code changes. This complexity can only be minimized by a conscious plan which anticipates change.

The final aspect which sets HEP computing apart from many other software projects is that it deals primarily with distinct events. Traditionally, the data structure has paralleled the event structure, possibly a tree with linked sub-structures, but storing the event as a single entity. But other methods of organization may be more efficient, for example column-wise ntuples allow better single-attribute access. We have seen that in the different phases of HEP computing we may wish a different granularity of access to the event information. Hence it is desirable to have a data model adaptable to changing requirements. The next section will further explore the relationship between event structure and the desire to plan for change.

4 Object Identification

In this section we will explore an example illustrating how planning for change affects a design decision. In this case, we will consider an object data model for HEP events. Our goal is to produce a structure allowing objects within an event to be flexibly linked yet capable of being stored and accessed from independent files.

More importantly, we'll want a structure adaptable to change.

In this model of linked objects, the most important item to understand is how the links should be represented. There are two methods by which a link can be created: by location or by identifier. The former simply tells the application to fetch some portion of information directly from a particular address. The address may be a real memory address, an offset in a record, or a complex structure combining a file name, record, and offset. An identifier simply adds a level of indirection by requiring the resolution of the identifier into a location. The resolution may occur via a database lookup, a simple table lookup, or perhaps a more complicated combination of actions.

Each method has its advantages. An identifier is more flexible, as the object to which it refers may be relocated without invalidating the link. Using a location as a link is much more efficient, and may be necessary if performance is critical.

The example is illustrated in Figure 1, which denotes a hypothetical data model composed of linked objects (hits, tracks, and electrons) which comprise an event, then further collections of events, runs, and lists of selected events. In Figure 1 each column is meant to represent an individual (collection of) file(s), for example all hits from all events would be stored together.

Also represented in the figure are the various links, with those implemented by identifier shown as bold lines and those implemented by location shown as thin lines. The goal is to have links between files be flexible, and hence to have such implemented by identifier. In this example each object type has a header (or metafile) containing a 'database' which resolves identifiers. The structure of the data requires some link between all of the object headers; in this example the link is represented as a location pointer, but it could equally well be simply the name of the file containing the metadata. Within a file the links can be direct location pointers.

To illustrate the flexibility of this toy model, consider the case where one wishes to concatenate the track and hit files to include only events with electrons. In this instance one would rebuild the track and hit files with the selected information. The event header would need to be updated to correct the pointers to the track and hit headers, but otherwise all of the identifier links would remain valid. In particular, the electron file could remain unchanged. This particular exercise then illustrates how object data might be migrated to a backup media but allow components of selected events to remain on local storage without the need for copying the complete event.

In this example we've looked at how planning for change could affect our data model, and specifically how it could affect the choices of implementation for links between objects. The data model can be flexible by adding or removing object types, and it can be flexible by restructuring the data so as to allow distinct storage options for objects within selected events.

5 Conclusions

In this paper we've examined the general problems facing HEP software development. A theme throughout is that software must be developed with expectations of change built in, including changes in goals, changes in methods, and changes in

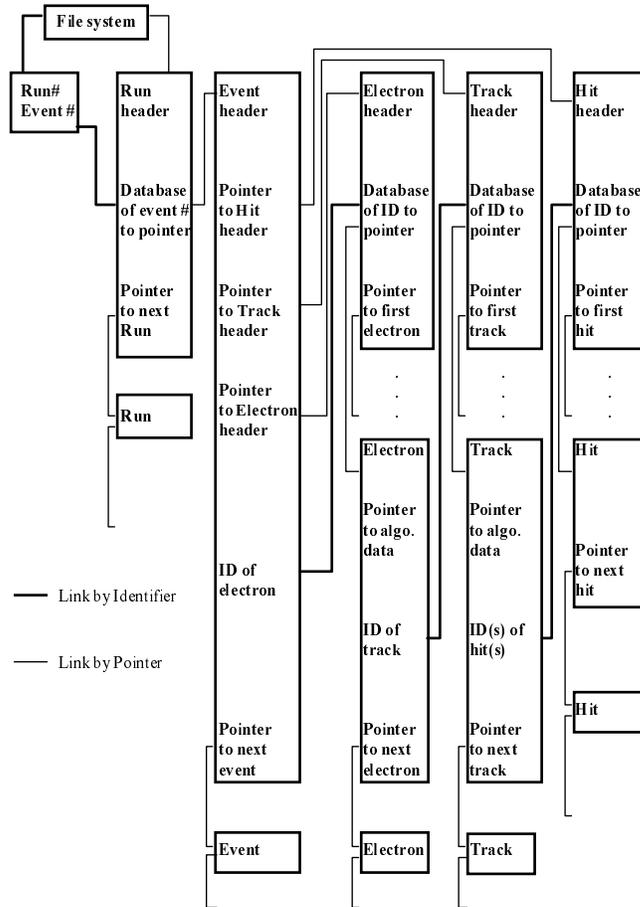


Figure 1: Identifiers as links

structure of both the code and the data. An example has shown how a data model could be designed to incorporate features more adaptable to changing data storage requirements.

CDF and DØ will be settling on future computing schemes within the next few months. The experiments will be working together closely in hope of finding common components. As these problems exist throughout HEP, it is hoped that our experience aids in future efforts to produce solutions with even wider HEP acceptance.