A Model for Computing at the SSC

Drew Baden
Department of Physics, University of Maryland

Robert Grossman
Laboratory for Advanced Computing
Department of Mathematics, Statistics, and Computer Science
University of Illinois, Chicago

June 6, 1990

Abstract

High energy physics (HEP) experiments at the SSC will show a substantial increase in complexity and cost over existing forefront experiments, and computing needs may no longer be met via simple extrapolations from the previous experiments. We propose a model for computing at the SSC based on technologies common in private industry involving both hardware and software.

Contents

List of Tables

1 Introduction

High energy physicists have traditionally had a schizophrenic attitude towards computer hardware and software. Demand for increases in computer performance (CPU, I/O, data acquisition, *etc.*) in high energy physics (HEP) has consistently increased over time, and contributes to some degree to the evolution of computers. On the other hand, the language of choice has always been fortran (considered extremely cumbersome for some applications), physicists in general are reluctant to take the time to learn how the computer really works, software is considered an unworthy thing to work on, and so on. There is justification for some of these attitudes. In order to be effective, the physicist must be very comfortable with the computer and be able to devote all of his/her attention to the problem he or she is trying to solve. However, the increasing demands the physicist makes (and will continue to make at the SSC) in computing may necessitate a basic change in the way things are done. We have a novel opportunity to review computing at the SSC for two reasons:

1. There are no archaic structures already set up (which are a basic cause of computer-conservatism, or inertia) and
2. The amount of data both online and offline anticipated at the SSC may already be too much to handle with present "classical" HEP computing techniques.

This paper outlines a scheme for computing at the SSC with an emphasis on data analysis. Such computing must incorporate state-of-the-art computer science tailored to the needs of HEP.

2 HEP: Computing

Data analysis in HEP has traditionally consisted of the following:

- **ONLINE**: Collect data, write to tape.
- **PRODUCTION**: Create "Data Summary Tapes" (DSTs) by reading from tape and running "production" code(s) which reconstruct data into dimensional quantities. Results are written to some medium, usually magnetic tape.

- **ANALYSIS** (of DSTs): This stage consists of performing calculations and making cuts based on either calculation results or variables already present in the record. The entire record (event) is read in for each event analysis.

Historically, each time an experiment increases in complication, the weak parts of the above outline have traditionally been dealt with though a logical extension of traditional techniques incorporating more technology. It is possible that this can continue, given the impressive technological advances in computer hardware present and anticipated. However, the future (SSC) will most likely result in an order of magnitude increase in the size of each event (relative to the largest ongoing HEP experiments to date), the number of events needed for analysis, and in the number of physicists who will be placing demands on a particular system (the size of the SDC is expected to approach $\simeq$ 1000 by the time data is collected). This amounts to a large, non-linear increase in the computing requirements. The question is, can we continue to throw all available increases in technology using the traditional techniques at our computing problems? If not, can we take advantage of both the large lead times now available before the data is collected and the tremendous increases in data storage, CPU performance, and software designs either present or anticipated to redefine the way we do computing such that we can hope that technological developments will help us solve our problems? The safest thing to do is to anticipate that we will have to do a bit of rethinking about the way we do computing. This means that we will first have to educate ourselves about what is currently available in the way of computer science (software and organization/management of resources) which might be applicable to HEP needs. In order to do this, we must develop a rational scheme, taking into account ramifications on all of the above three categories, **ONLINE**, **PRODUCTION**, and **ANALYSIS**.

2.1 ONLINE

During the 1988-1989 Tevatron run the CDF experiment wrote 80-100 kbyte event records to 9 track tape at the rate of about 1 Hz. Each tape can hold approximately 120 Mbytes at 6250 bpi, and with a 20% safety factor this limited the tapes to about 1000 events each. At 1 Hz, a tape was filled every 15 minutes.

Near the end of the run, the data was stored on 8mm cassettes, which can hold approximately 2.3 Gbytes, a factor of about 20 greater than 9 track tapes. The 8mm drives can write data about 250 kbyte/sec. This seems satisfactory for CDF needs, providing that the product of the event size times the event rate never exceeds #tapes×250 kbyte/sec given today's technology. EXABYTE (the 8mm of choice at FNAL) has just started to ship the next generation of 8mm drives. Improvements include:

- Factor of 2 increase in bandwidth to 500kbyte/sec
- Factor of 2 increase in capacity to 4.6 Gbytes

However, 8mm tapes can only be seen as a solution to long-term storage problems. Access to data is quite limited on an interactive basis, since the device is not random-access. A simple calculation based on the above numbers says that it takes $\simeq 2.5$ hours to read an entire tape. This is not as much of a problem for the **ONLINE** storage as it is for the **PRODUCTION** and **ANALYSIS**, however it does not make sense to support too many different storage media at SSCL. In Texas, 8mm tapes will probably not be sufficient. Anticipated event sizes of the order of 1 Mbyte would require 20 8mm tapes online to get a rate of 10 Hz (given the writing speed is 500 kbyte/sec). (Note: These tapes are guaranteed to write with an error rate of less than 1 in 10^{15} bits and read with an error rate of less than 1 in 10^{13} bits. Given a 1 Mbyte event size, this corresponds to a write error of less than 1 in 125 million events and a read error of less than 1 in 1.25 million events.)

There are several alternatives to magnetic tape. In the following sections we present a very brief exploration of data storage alternatives (relative to magnetic, 8mm tapes).

2.1.1 Data Storage Media

- Optical Disks:

  At the SSC, optical disks will have to be considered. Such devices are relatively new on the market, but have already reached milestones in performance worth considering. Write-once-read-many (WORM) drives have been available for several years from various manufacturers.

Characteristics	KODAK	SUMMUS
Bus	SCSI,IPI-3	Unibus,Q-BUS,VME,SCSI
Capacity	6.8 Gbyte (13.2MBlocks)	594 Mbyte (1.2MBlocks)
Volume (in^3)	3000	150
Weight	27 kg	3 kg
Transfer Rate	1 Mbyte/sec	680 kbyte/sec
Access Time	700 ms (Data within disk)	90 ms (average seek)
Price DRIVE	$20,000	$6,000
Price DISK	$700	$250

Table 1: Comparison between Kodak's 6800 and Summus's LightDisk

In late 1988, erasable optical drives started to become available. Although slower than conventional magnetic disks, erasable optical drives have the advantage of a larger information density (currently a factor of 2-10 over magnetic) and removable media.

For example, KODAK makes a device called the "OPTICAL DISK SYSTEM 6800" and SUMMUS now markets a similar device called the "LIGHTDISK SO-600". The KODAK device has a larger capacity but is slower and more expensive than the SUMMUS device. See table 2.1.1 for a comparison.

The KODAK device can be purchased with a robotic library server (called a "jukebox") which holds 250 disk with an advertised access time of 6.5 seconds per disk costing from $100,000 to $200,000. SUMMUS has not announced such a product, but it will probably be available some time.

- Optical Tape: CREO PRODUCTS (Vancouver, British Columbia) has announced the availability some time this year of a new optical tape drive, using the so-called "optical paper" development by ICI. Honeywell Test Instruments Division has reportedly agreed to market the drive to end users. The characteristics of this drive are

  - 12 inch diameter tapes, 1/2 inch wide (similar to conventional 9-track tapes) and 2400 feet long

- 1 TByte capacity per tape
- 3 Mbyte/sec transfer rate
- 28 sec average access time to any byte on any tape, with 60 sec. end-to-end
- $225,000 per drive
- $5,000-10,000 per tape
- Write-once media.

One of the biggest advantages of having such a medium comes from its extremely large capacity. A Tbyte is approximately 10^6 events. This entails 100 tapes to store an eventual $\simeq 10^8$ events, which means a rather large cost in tapes given the above price. Should the price drop a factor of 10, it would no longer be a consideration. However, this solution is basically just a technological extension of 8mm tapes in that it is really not a device that one could use for interactive computing. This issue is discussed further below.

- RAID

  Called **R**edundant **A**rray of **I**nexpensive **D**isks, RAID is a scheme which uses a farm of small disks with multiple parallel data paths in order to simulate the far higher performance of a large disk farm. In the limit of an infinite number of disks and data paths, the fastest way to write data is to write 1 bit per disk. The RAID project uses block transfers of order $\simeq$ 4kbits writing over parallel links to 500 Mbyte disks, and is used by multi-processor farms (connection machine-type computers). It is worth investigating, but the small capacity compared to optical disks means that a great number of disk drives will have to be purchased and managed. However, let's assume that in $\simeq$ 5 years time these disks will have a 1 Gbyte capacity, 1 Mbyte/sec bandwidth, and $1000 price tag. If events are written at 100 Hz and are 1 Mbyte long, then at least 200 disks would be required for online purposes. To store 10^7 events of 1Mbyte length, we would need 10000 disks at a cost

Device	Capacity (1 MB events)	Write Speed (events/sec)	Cost (10^7 events)	Interactive?
Optical Disk (Kodak)	6800	1	$\simeq \$1.2M$	YES
8mm tape (Summus)	4600	.5	$\simeq \$0.2M$	NO
Optical Tape (Creo)	1,000,000	3	$\simeq \$1.0M$	NO
RAID	10,000,000	≥ 100	$\simeq \$10.0M$	YES

Table 2: Comparison between various optical disk, optical tape, and 8mm magnetic tape, and magnetic disks for online and offline storage. Event size is assumed to be 1Mbyte and number of events stored is estimated to be 10^7 for the first year of running.

of $\simeq$ \$10M. This would probably eliminate this as a choice for mass storage, but may play a role in caching.

2.1.2 Online Storage Proposal

Table 2.1.2 offers a comparison between the SUMMUS 8mm Gigatape (today's standard) using the new EXABYTE specifications (.5Mbyte/sec, 4.6Gbyte storage), the KODAK read/write optical disk (as an alternative device which has random access capabilities) and the CREO optical tape (for data storage). Events are assumed to be 1Mbyte long, and it is assumed that $\simeq 10^7$ events will be collected during the first year's running.

Given the online specifications of an SSC experiment (assuming 1 Mbyte events, 10 Hz to tape), a proposal for online storage using the KODAK device would entail:

- Drives: At 10 Hz, we would need 10 of the KODAK-type optical disk drives. The cost of this before university discount will be on the order of $\simeq \$200,000$. Each drive's disk would have to be changed every $\simeq 2$ hours, requiring a disk change somewhere every $\simeq 12$ minutes.

- Disks: Each disk holds 6800 events. A total of $\simeq 10^7$ events means we will need $\simeq 1500$ disks for the online. The cost of this (at $\simeq \$700$/disk) is $\simeq \$1,000,000$ before university discount.

One can assume that these prices are upper limits to the eventual real cost at the time the experiments are running.

Note that if the data taking were to begin today, these costs would in no way be prohibitive relative to a cost of $\simeq \$500M$ for each SSC experiment. However, the real power of using random access devices for mass storage is realized in the advantages for the other two categories of HEP computing, **PRODUCTION** and **ANALYSIS**, as outlined below. It is not *necessary* that data be written online onto a random access device except that it will be desirable to limit the different number of storage for support reasons to as few as possible, and given that one will want to minimize the number of times one is copying such large amounts of data from one device to another.

2.2 PRODUCTION (of DSTs)

There are 2 scenarios to consider for how offline production will proceed:

1. High-level objects (electrons, muons, jets, photons, *etc.*) are "produced" offline using special-purpose codes operating directly on the raw data. The program will use calibration constants which are calculated using the raw data itself after (or as) it is written.

2. Production will be done in the last trigger level (Level 2) of the DAQ system. This is possible only if the final constants are available in real time and there is enough computing power available.

The choice will depend on the type of hardware which one will be using in the detector. The safest choice would be to prepare for item 1. above, given the kind of hardware which people have been thinking about lately.

2.2.1 Proposal

Assuming case 1. above, one could imagine that a farm of processors, with access to the database of constants and the data via an optical "juke-box" (see section 2.1.1 above) would suffice. Note that unlike 8mm tapes, there is no need for staging. The numbers look something like this:

- On the order of 2 "juke-boxes", one for providing data to be read by the production program and one for storing the resultant DSTs. Each "juke-box" is priced at $\simeq \$100,000$ to $\$200,000$. Total cost is no more than $\simeq \$400,000$ at today's prices before university discount.

- The second "juke-box" will require additional optical disks (relative to the ones used to record the data online). One can probably come fairly close to reality by estimating that the number of disks used by DSTs is no larger than $\simeq 2\times$ that used for the raw data (assuming an expansion of 2 and no reduction in the number of events). The cost of this, again at today's prices before discount, is $\simeq \$1,000,000$ for 10^7 events. To speed up access to the DSTs, the juke box of optical disks should be provided with magnetic disk caching. Hybrid systems of this type are now becoming commercially available. Such systems could be used off the shelf or higher performance and more specialized caching systems could be designed and developed expressly for this purpose.

2.3 ANALYSIS

Traditionally, each new HEP experiment has invented it's own software "system". Physicists then use this system for **PRODUCTION** and **ANALYSIS**. This has always been simply a matter of convenience. At the SSC, such "convenience" may never appear given $\simeq$ 1000 physicists and $\simeq 10^7$ events with $\simeq$ 1Mbyte per event, and one has to consider a different "system" for **PRODUCTION** and **ANALYSIS**. Experience at CDF reinforces this for a number of reasons:

- Linking. Most CDF analysis is Vax-based. The offline software packages are quite large, and even with shareable (and shadowing) link times are not in the "interactive" ballpark range. Even if they were, the nature of the physicist ensures that (almost) no code freshly written

will ever run correctly the first time, which means that a non-trivial fraction of the computer resources are devoted to linking.

- Data. Data at CDF is written in a device-independent manner, necessitated by the device dependencies of the collaboration. This means one has to add structure to the data, and this increases it's length and complexity. Given that CDF is dependent on magnetic tape for mass storage and magnetic disks for interactive storage, data space for the individual user is at a premium.

Necessity being the mother of invention, most of the CDF physicists are now turning to the CERN-written PAW program to produce physics results. This program allows interactive data analysis (much like a spread-sheet program) and incorporates an inline fortran interpreter which completely eliminates the need for linking. The interpreter has been found to be fast enough, since most of the time spent when running in the interpreter mode is for I/O. The scheme for data analysis looks like this:

- Binary files (device-dependent, unformatted I/O via very simple fortran WRITE(LUN) x,y,z...) are made from the DSTs. These files contain only specific parts of each event necessary to do the particular analysis as determined by each physicist. The files are created by running a single job using the **PRODUCTION** "system", and the data are stored on disk as reals and integers as appropriate. These files are extremely small relative to the DSTs (factor of at least 100) and quite easy to keep track of.

- HBOOK4 (PAW) files are created by running a very simple interactive program which reads the binary data and fills "ntuples" (a matrix where the rows are events and the columns are physics quantities). There is very little to this program other than straight fortran and some HBOOK commands with the proviso that one has to be very careful to keep track of the words in the binary file, since it is not self-describing. The PAW file consists of ZEBRA banks, which is not in any way compact, but can be remade quite quickly and easily provided the binary file is available. This (PAW) file is needed to run the PAW program.

- Run PAW. The program allows one to make histograms, scatter plots, fits (interactive MINUIT), and allows one to place cuts on the data and study correlations and anti-correlations interactively. PAW is not easy to use - the commands are "unix"-based and the documentation has been found to be less than clear - but the program is extremely flexible and powerful and has become quite popular at CDF

The popularity and success (given that CDF physics output is not unsatisfactory) of the PAW program warrants consideration at the SSC. However, the dependency on binary files is bothersome for a number of reasons (e.g. files are device dependent and unstructured hence not easily self-describing), and the file structure of the PAW file is very primitive. It is conceptually straight forward to update the concept.

2.3.1 Data Structure

One can easily improve on the above "PAW" scenario in such a way as to minimize the waste of resources. First, consider the physicist who is either resident at SSCL or can log into a central computer system over a high speed link (the kind of T1 lines which are proliferating now). Given the above proposal to write the data to optical disks (see section 2.2.1) large amounts of data are accessible at the SSCL without having to go to tape (the "juke-box" allows access to about 1Tbyte of data, or about 10^6 events on disk). Given the belief that it will be I/O and not CPU which will limit us, we can consider ways of restructuring the way data is stored and analyzed so as to minimize the I/O. If one accepts that there may be a benefit in separating computing in HEP into **PRODUCTION** and **ANALYSIS** problems, then any and all solutions to (software) computing issues will have to start with the way the data is structured. Data structures should be designed to maximize the benefits of the increasing technology - the two are not independent. Intelligent data structures could greatly aid in bookkeeping efforts during production as well as reduce the terrific amount of I/O required for data analysis.

Let's look at the "classical" analysis situation. Say I have a sample of DST events (therefore these events have already been run through a production program) and I want to look for $t\bar{t}$ (TOP) candidates in the $e + \mu$ channel. I write a (fortran) program, link it, and run it. Here I am assuming that

these events are approximately 1 Mbyte long. The following is a play-by-play description of what happens.

1. Read in each event. If the events are on tape, access the tape first. For each event
   - Cut on global event quantities (e.g. primary vertex, trigger bit, *etc.*).
   - Require at least 1 e and 1 μ candidate.
   - Cut on e and μ quantities to get a "clean" sample.
   - Global event cuts (jets, $\not\!\!E_t$, *etc.*).
   - Plot p_{te} v. $p_{t\mu}$.
2. If desired, save events passing e and μ p_t cuts onto disk or tape.

This procedure is very straight-forward, and there are no obvious inefficiencies to be trimmed away. However, there are inefficiencies which involve the repetition of this procedure many times due to for example debugging the code or the application of additional cuts which result in smaller and smaller data sets. Events with a record size of 1Mbyte (or more) can easily result in an unacceptable I/O load on the system, necessitating either a drastic reduction in the size of each event through either a compression or trimming (miniDSTs, μDSTs, *etc.*) and/or a drastic cut in the number of events.

An alternative to the "classical" way of storing data in event records consists of actually adding to the record length by introducing a structure to the data which would make it possible to read in only parts of the data record as needed. In other words, turn the data into a real "object-oriented" database. In the following sections we present a layperson's explanation of a relational and object-oriented database and how it may benefit the analysis of HEP data.

2.3.2 The Relational Database

The relational database was developed in the early 1970's, sparked by the influential paper of Codd[1]. Prototypes emerged in the middle to the late

[1]E. Codd, *A relational model of data for large shared data banks,* Com. ACM, **13–6**, pp. 377-387.

Run	Event	Pte (p_t e^+/e^-)	Ptmu (p_t $\mu^+\mu^-$)
1001	126390	74.1	3.5
1006	69372	0	78.0
1007	574930	62.3	215.2
1020	6320	99.2	0
	⋮		

Figure 1: A "typical" HEP table called **HEPdata**.

1970's, including Ingres[2] and System R[3]. To visualize a **relational database**, consider a set of variables, or equivalently a multi-dimensional system space. In database parlance, each dimension (or variable) is referred to as a **domain**. Now consider that in this system, there are relationships, or conditions, between domains which tend to group certain values of each domain across domain boundaries. Such relationships group the domains into tables where the columns are domains and the rows (called **tuples**) are the grouping of domains. As an example, consider a HEP event at the high level where one is concerned with "objects" such as the 4-vectors of the electrons, muons, jets, and *etc.*. One domain is the set of all electron transverse energies, another might be the same for muons. Each row is an event, which is the *relation* which causes particular electrons and muons to be grouped, and the entire set of relations forms a **table** (or **relation**) in database parlance. In a relational database, the number columns is fixed. See figure 1 for an example of a HEP table.

An important attribute of a relation is that it abstracts a certain type of file. A file of this type would be a sequence of records, one for each tuple in the database. Each record would consist of a sequence of fields, one for each column in the table. The definition of a relation implies that all records in the file would have the same number of fields, that no duplicate records be allowed, and that each field be atomic and have no additional structure.

Relational databases support special languages (called *query languages*)

[2]Michael Stonebreaker, ed., **The Ingress Papers,** Addison–Wesley, 1986.

[3]Astrahan, et. al., "System R: A relational approach to database management," *ACM Transactions on Database Systems,* 1 (1976).

Range of t is HEPData Retrieve into TOPTable (t.Run, t.Event) Where t.Pte> p_{cut} and t.Ptmu> p_{cut}

Figure 2: A typical query.

Run	Event	Pte (P_t e^+/e^-)	Ptmu (p_t $\mu^+\mu^-$)
1007	574930	62.3	215.2
	⋮		

Figure 3: Result of query on HEPdata.

which provide for the analysis of the data in the database. For instance, in the HEP example, if you wanted to collect all events with an electron AND a muon with p_t above a cut (p_{cut}), the query language may take the form as in figure 2 and produces a *new* table as in figure 3. A query is simply a function on the database acting on relations and producing other relations. It is an essential feature of such queries that they produce other relations, which can then be queried in exactly the same way. Relational databases also support the taking of intersections and unions of relations, which are called joins and meets, to produce other relations. In this way, a relational data base provides a powerful means of completing queries on large data bases containing containing numeric and alphabetic fields.

2.3.3 The Extensible Database

By the mid 80's, it was recognized as a good idea to relax the requirement that domains consist of atomic objects and extend databases to support a collection of data types richer than merely numbers and strings. By the late

Range of t is HEPData
Retrieve into TOPTable (t.Run, t.Event)
Where Number(TopCandidates(t.Run)) ≥ 0

Run	Event
1007	574930
	⋮

Figure 4: A query and the relation it produces.

80's, a number of such systems have been proto-typed, including Iris[4] Orion[5] and GemStone[6]. Imagine a relational database which is extended to support not only numbers and characters but also physical (and logical) objects such as events, muons, calorimetry tower lists, and *etc.* In other words, the table entries are no longer assumed to atomic, but rather to have an internal structure. For example, in figure 1, imagine that there is a column (domain) called "TOP" which is a basic data type known to the system. Queries such as in Figure 4 would then be possible.
It is important to note that

- queries always produce other tables, which can themselves be made the subjects of queries;
- when queries produce tables, the underlying data itself are not copied, but rather appropriate pointers are introduced pointing to the original

[4]D. Fishman et. al., "Overview of the Iris DBMS," *Object-Oriented Concepts, Databases, and Applications,* W. Kim and F. H. Lochovsky, editors, ACM, New York, 1989, pp. 219–250.

[5]W. Kim, N. Ballou, H-T. Chou, J. F. Garza, and D. Woelk, "Features of the ORION object-oriented database system," *Object-Oriented Concepts, Databases, and Applications,* W. Kim and F. H. Lochovsky, editors, ACM, New York, 1989, pp. 251–282.

[6]R. Bertl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams, "The GemStone Data Management System," *Object-Oriented Concepts, Databases, and Applications,* W. Kim and F. H. Lochovsky, editors, ACM, New York, 1989, pp. 283–308.

objects, which may be quite large;

- indices can be introduced to speed up subsequent queries.

The inclusion of new data types requires that new query operators be supported, new storage and access methods be developed, and new methods of optimizing queries be found. For example, it is important that the database support the storage and retrieval of events of an arbitrarily large size. It is also important that query operators defined on entire objects be supported, such as operators returning various statistical quantities. It is essential for HEP applications that arbitrary Fortran (and C) user-written subprograms interface easily into the query language, *i.e.*that there be a "hook" which the user can use to input customized operators to perform the queries. This is simply a logical extension of existing traditional HEP analysis methodology.

At this time, there is no consensus about how objects should be incorporated in a relational database. One viewpoint is to change the relational model as little as possible and build a complete extensible database on top of it. For example, Stonebreaker[7] describes a mechanism for a user to register new abstract data types into the University version of Ingres. Because of the built-in access methods used by Ingres, the new data types must occupy a fixed amount of space. Still another viewpoint is to provide a modular and modifiable system on top of which extensible databases for specific applications can be built. The Exodus database described by Carey[8] is an example of such an approach. See Carey[9] for a recent survey covering some of these issues and detailing other approaches.

Also note that although object oriented programming is closely related to object oriented databases, they are in fact two different concepts.

2.3.4 The Scientific Database

Beginning in the late 70's and early 80's, there has been increasing interest in statistical and scientific databases. These types of databases differ in a

[7] M. Stonebreaker, "Inclusion of New Types in Relational Data Base Systems," *Proceedings of IEEE/Data Engineering,* IEEE, 1986, 262–269.

[8] M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson, E. J. Shekita, "The Architecture of the EXODUS Extensible DBMS," *Proceedings of the Object–Oriented Database Workshop,* ACM, 1986, 52–65.

[9] "Special Issue on Extensible Database Systems," *Database Engineering,* 1987.

number of important ways from conventional relational databases.

- The datatypes are different: datatypes in a statistical database include time series, matrices, and usually make provisions for missing data, outliers, etc.; datatypes in a conventional relational database are usually limited to numbers and strings.

- Operators computing averages, standard deviations, regressions, etc. are important in a statistical database, while operators computing meets and joins are important in a conventional relational database.

- Updating and modifying the data in a conventional relational database is frequent; on the other hand, in a statistical database, data may be added to the database, but is rarely changed.

Much of the data in a statistical or scientific database is static: updates are infrequent, but the queries are usually computationally very intensive. For example, the queries of interest typically cannot be answered from the stored relations and data; rather, some information typically needs to be computed, such as a regression. In other words, a typical query requires extracting features or some type of summary information from the database. In the past few years, there has been increasing interest in statistical and scientific databases. See for example, Hammond[10] and Rafanelli[11] and the references cited there.

2.3.5 A HEP Database

The analysis of HEP experiments would benefit from an object oriented extensible database, supporting basic objects such as runs, events, candidates, etc. and allowing statistical and graphical queries.

The database would support *tuples* of objects, *relations* (expressing the relationships between tuples), and *operators* performed on the tuples and relations. The objects would include events, candidates, electrons, muons,

[10]R. Hammond and J. L. McCarthy, editors, *Proceedings of the Second International Workshop on Statistical Database Management,* Springer, 1983.

[11]Maurizio Rafanelli, "Research Topics in Statistical and Scientific Database Management," in *Statistical and Scientific Database Management,* M. Rafanelli, J. C. Klensin, and P. Svensson, (editors), Springer, Berlin, 1989.

jets, photons, etc. A relation could be viewed in many different ways: as a table, as a histogram, as a graph, or as an icon.

The operators would include:

Tabulate This would display the relation in a tabular format, not unlike a spreadsheet.

Graph This would graph the data, in any of a variety of formats.

Histogram This would provide various histograms associated with the tuples in a relation.

Iconize This would collapse the relation into an icon so that it could be moving around and so that other statistical operations could be performed on it.

Group This would allow the structure of the relations in a database, similar to the way files are structured into directories or prose is structured in an outline. That is collection of similar relations could be gathered together and collapsed into a single entity, until a later time that the underlying relations need to be accessed.

Mean This would compute the mean and other standard statistical functions of a relation.

Function fitting This would allow fitting functions to describe the functional relation of the data specified by tuples, regardless of the view of the data. That is, regardless of whether the data was displayed in a graphical, histogram, or iconic format.

Such a database should be developed with the following guidelines:

1. It should be designed using principles of object oriented programming and coded in C^{++}.

2. It would be developed in a Unix environment and run under a graphics standard such as X-windows.

3. It should be distributed, allowing for data to be distributed on a number of networked data servers, and queries to be processed on a number of network query servers.

4. It should allow for user-written (perhaps even fortran?) "subroutines" to perform customized queries.

In a system with an object-oriented data structure, magnetic and/or optical tapes (see section 2.1.1) would be used for archiving purposes only, with all data and/or pointers to the data stored on random accessed devices such as optical disks. The advantage of disk over magnetic tape is in the ability to have a file structure. Access to files on disks which contain more than one file does not mean reading each preceeding file. The organization of the physical data can be optimized to limit the amount of "unused" bits read in on each query (so-called query optimization). For instance, each domain can be structured like a file, and read in as needed. For instance, each domain can be structured like a file, and so only information (domains) which are needed for the query are read in. Another possibility is that frequently used collection of events (*e.g.* inclusive electrons, inclusive jets, *etc.*) can be processed into a table which contains pointers to a subset of all possible domains. Subsequent queries would be on these subsets of events. For **PRODUCTION** purposes, the entire collection of objects would be read into memory. Note that this software would not be hardware independent.

With the data in an object-oriented database, the above $t\bar{t}$ $e\mu$ analysis is effectively reduced to a single query similar to figure 2. This query would perform the following:

1. Retrieve for each event from a given file (table) information needed to calculate the number of electron and muon candidates.

2. Require at least 1 of these (passing cuts) and

3. Save the *list* of events.

Once the list is saved, one could go back to the data file and read in global event "objects" for events in the list, make cuts, save the list of events passing cuts, and *etc.* Advantages of this scheme are

- The fact that one can now use the hardware and system software to modularize the analysis could significantly reduce the total amount of I/O.

- A powerful bookkeeping system could be set up by adding a "bookkeeping" object during **PRODUCTION**. The object could be accessed independent of the rest of the event to keep track of event numbers, topology, *etc.*

- Any subset of events could be copied onto disk from a list without sequential access of all events (since now an "event" is like a VAX "file" with a directory you can use to find it) and mailed to a collaborating institution.

- With the help of a "PAW-like" analysis program which is written for this particular data structure, access to the data would be immediate and interactive, and analysis would consist of writing "command" files and "queries" to drive the "PAW-like" program.

2.3.6 Building a HEP Database

In the last section, we outlined some of the basic requirements of a HEP database. In this section, we discuss some of the issues involved in designing and building a HEP database. We feel that the main issues will be the following.

- A high performance HEP requires finding the right balance between primary (magnetic memory), secondary (magnetic and online optical disks); and tertiary (nearly online optical tapes and disks) storage devices. Performance can be improved by caching secondary storage in primary storage, and tertiary storage in secondary storage; and by using data indices residing in memory to speed up the retrieval of data. Although these are standard database techniques, they will have to be adapted and tuned to fit the needs of a HEP.

- A high performance HEP requires balancing data transfer and cpu requirements of the database. To process data queries at acceptable levels requires queries be processed in a parallel or distributed fashion. Imagine that a one terabyte database is divided into 100 buckets of 10 gigabytes each and that a $10,000 RISC workstation was assigned to process queries for each bucket. Alternately, imagine that the 1 terabyte database is divided into 10 buckets of 100 gigabytes each and

that $100,000 query processor is assigned to each. Most queries could be arranged so that each query processor could work independently, with only minimal communication required between them. Given these assumptions, a distributed query processor of the type just described would result in a nearly linear speed up in the queries. The hardware cost of such a system would be approximately 1 million dollars. It is essential that a HEP database be designed to support distributed query processing as well as distributed access to the data.

- Although there are many commercial databases available, it seems unlikely that any of them are suitable for a HEP. More likely, a HEP will incorporate various database tools and utilities as they become available.

In the remainder of this section, we discuss the last issue in more detail. There are several ways of building a HEP database:

1. build it on top of a conventional commercial relational database
2. build it on top of a commercial high performance relational database
3. build it on top of an object oriented commercial database
4. build it using database tools and utilities
5. build a HEP from the bottom up

There are several commercial databases available, including Sybase, Ingress, and Oracle. Option 1 would build the HEP on top of one these databases. These databases do not support the basic data types and operators required of a HEP; moreover, they do not scale to allow for 1–10 Terabyte database imagined for the HEP. In general, they do not seem suitable for large, high performance scientific databases.

Option 3 would build the HEP on top of a commercial object oriented extensible database. At present, such databases all allow for the creation of very general object types. This introduces performance penalties that are unacceptable for the HEP. Also, since these databases are just coming to the market now, it is too early to commit to a company that may or may not be around six months from now, and to software that only represents a few man years of work.

Option 4 would build the HEP using tools and utilities designed to help shorten the cycle of designing and building a database. The main product of this type is Sun Microsystem's NETISAM. The idea would be to use whatever tools and utilities are available, but in the end to produce an extensible, object oriented database that supports the data types and operators required for a HEP database.

Option 5 would design and build the HEP database from the bottom up. Such an approach would produce a database with the highest performance, but would make it more difficult to modify and maintain the code.

At present, it is too early to say with certainty which approach is the best, except that Option 1 is not viable. Our recommendation is to prototype Options 4 and 5 over the next 1 to 2 years and at the same time to monitor closely Options 2 and 3. It is possible that one of the latter two options could become viable within this time frame.

3 Remote Access: Client/Server

It is probably universally agreed that the large SSC detector collaborations will want to have a computing group resident in some central location, and the traditional (and logical) choice will be the SSCL. In a scheme for computing as suggested above, access to the data by physicists resident at SSCL would be trivial. In the next section, we discuss the more non-trivial access, by non-resident collaborating members over network links.

3.1 Collaborating University/Laboratory Access

Access to the data from outside SSCL will present a real challenge to the implementation of any computing system. It will probably not be sufficient to require all data be accessed locally at SSCL while at the same time it will not be possible to furnish every collaborating institution with all of the data (certainly very little, if any, raw data will be distributed as a rule). It is envisioned that a reasonable compromise will be to allow both. Access to the entire data set at SSCL should be made possible through high bandwidth, reliable links directly into SSCL. Also, a scheme with "satellite" centers distributed uniformly (in bandwidth space) throughout the community (several per collaboration perhaps?) should be considered based on avail-

able resources. Access to subsets of the data (high level objects) should be via distribution of specific subsets of the data to collaborating institutions. For instance, if the optical disk is chosen as the primary storage medium, a second "juke-box" can be used to produce disk copies in batch at SSCL for outside institutions. Access to this data will require one of the disk drives to be locally resident. The $\simeq \$20,000$ cost is not out of the grasp of even the smallest collaborating institution. However, the $\simeq \$700$ per disk might become a problem. One can imagine that the technology will catch up to this between now and the next 8 years.

The above proposed scheme calls for turning the entire dataset into an object-oriented database. Software would have to be developed to do the queries on the database, store any resultant "objects", and *etc.* Analysis of the data would be through a scheme much like the one used in the "PAW" program (as opposed to the traditional approach of writing, linking, and running code). The trend in private industry appears to be in the direction of distributed computing, using "clients" (local workstations) for interactive needs (e.g. make histograms and mathematical fits) and "servers" (host computers with access to the data) sending data over high-speed links. Such a scheme, although attractive in many ways, depends *entirely* on the reliability of the data links. For this reason, consideration should be given to providing sufficient redundancy through the "satellite" centers (see above). Note that the client/server approach has already been standardized in the MIT program "X-WINDOWS" system, and MICROSOFT has recently committed itself to this direction.

4 SSCL Implications

4.1 SSCL Computer System

Two of the largest HEP computing centers in the U.S., SLAC and FNAL, are interesting examples of two very different ways to solve the same problem. SLAC relies on powerful, well-maintained and supported mainframes available to all experimenters. At FNAL, on the other hand, much (although certainly not all) of the computing is done at collaborating institutions, an informal and primitive "distributed computing" arrangement involving many

different machine architectures and operating systems. One of the more pressing issues the new SSCL faces concerns this issue of central-mainframe(s) v. distributed computing (workstations). In the above scheme, however, both will have to be used. One needs a large central computing center with sufficient I/O bandwidth organized around the large amounts of data. These computer(s) would act as servers to a farm of clients for **PRODUCTION** of DSTs and for CPU intensive simulation work as well as servers to any client (workstation) at a collaborating institution for remote access and analysis. (Note that such client (workstations) will invariably have enough CPU power to handle most analysis requirements if one assumes that a sub-$10,000 workstation in the 10-20 MIP range would be readily available.) The proper segmentation of such a computer system around I/O, serving, and cpu-intensive tasks should be studied. One would suspect that there is a maximum efficiency for some form of segmentation, providing that the specifications of the computer system are well established.

4.2 Existing Resources

Another, perhaps more political, issue concerns the problem of what to do with the current tremendous investment in and variety of computing at the universities and national laboratories. A "PAW"-like system would help alleviate the issue. The idea of "operating system" may very well, at some point in the future, disappear.

4.3 Key Factors

The following is a list of key areas which are needed for the above computing scheme to be successful:

- Networks. The proliferation of T1 (or greater) cable links is good news. It will be extremely important that there be easy access to this network by any collaborating institution, and that the network be reliable. This may very well be the weakest link in principle, however it is clear that the importance of powerful networks has not been lost on the computer industry.

- Data Storage. No more tapes, except maybe for achival storage. Serious resources (money) should be spent on file oriented disks with the

goal that at least 1Tbyte of data be accessible interactively. Projected technological advances makes this a reasonable goal.

- Software. There is a large amount of code to write to set up such a system but the basics are already there (X-windows from MIT, Windows at Microsoft, *etc.*). An interactive "PAW"-like system would have the advantage that if it was supported in UNIX, VMS, VM, and MS/DOS, there would probably be no other operating system one would have to consider. (In 8 years the list could be even shorter than this.) The server/client scheme is reported to be operating-system independent to some degree. The "PAW"-like system would necessitate creating the following:

  - A "PAW"-like shell. In principle, there is no reason that the current PAW program could not be modified for this.
  - An extensible relational database to store, retrieve, and access the data, as described in section 2.3.1.
  - A database query language to connect "PAW" to the data. The design of an extensible, relational database together with the appropriate query language is a large task. There are presently several DOE and NASA funded software projects with similar aims, which could either be extended or absorbed if appropriate.

Data taking is scheduled to begin sometime during 1998. The detector builders claim that it will take at least this long to build the detector. In implementing the above scheme, one would not have 8 years of lead time since software systems will be needed as early as possible to set up a working collaboration. A powerful, modern system as just outlined could be functioning several years before data dating if enough resources are provided. And such a system could in principle be utilized by all of the collaborations which will do physics at SSCL, obviating the need for duplicating the work of setting up a different **ONLINE**, **PRODUCTION**, and **ANALYSIS** system for each detector. An additional justification can be made that such a system would be extremely powerful and available to both the scientific and business community - a beneficial HEP spinoff.