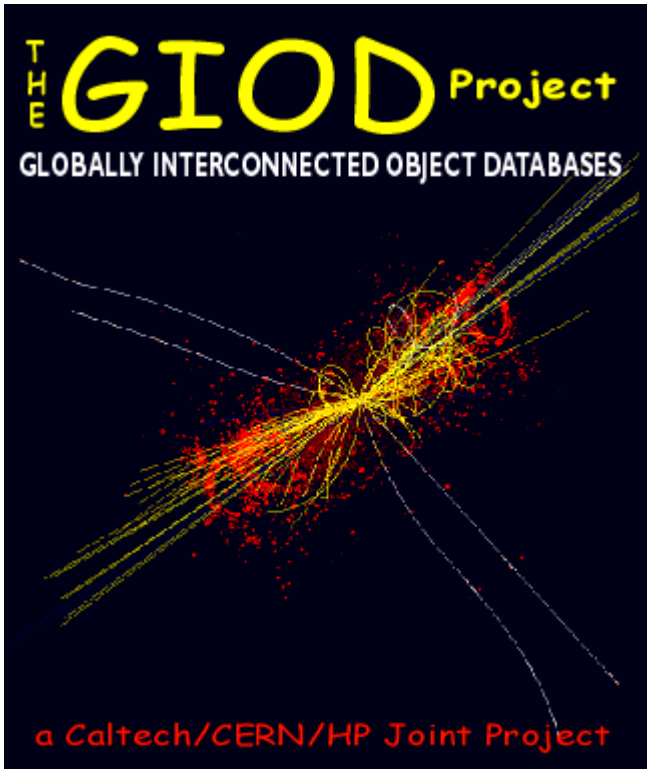
Final Report of the

Globally Interconnected Object Databases

(The GIOD Project)

A joint project between Caltech, CERN and Hewlett Packard



October, 1999

Julian J. Bunn, Harvey B. Newman, Richard P. Wilkinson
(Caltech, 1200 E. California Boulevard, Pasadena, California)

The GIOD Project: Globally Interconnected Object Databases for HEP

Introduction

In late 1996, Caltech HEP (H. Newman), Caltech's Center for Advanced Computing Research (P. Messina), CERN IT Division (J.Bunn, J.May, L.Robertson), and Hewlett Packard Corp. (P.Bemis, then HP Chief Scientist) initiated a joint project on "Globally Interconnected Object Databases", to address the key issues of wide area network-distributed data access and analysis for the next generation of high energy physics experiments.

The project was spurred by

- the advent of network-distributed Object Database Management Systems, whose architecture holds the promise of being scalable up to the multi-Petabyte range required by the LHC experiments
- the installation of large (0.2 TIPs) computing and data handling systems at CACR as of mid-1997,
- the fundamental need in the HEP community to prototype Object Oriented software, databases and mass storage systems, which are at the heart of the LHC and other (e.g. BaBar) major experiments' data analysis plans, and
- the availability of high speed networks, including ESnet, and the transatlantic link managed by the Caltech HEP group; as well as the next generation networks (CalREN-2 in California and Internet-2 nationwide) planned to come into operation in 1998-9 with speeds comparable to those to be used by HEP in the LHC era.

A plan to understand the characteristics, limitations, and strategies for efficient data access using these new technologies was formulated by Newman and Bunn in early 1997. A central element of the plan was the development of a prototype "Regional Center". This reflects the fact that both the CMS and ATLAS Computing Technical Proposals foresee the use of a handful of such centers in addition to the main center at CERN, with distributed database "federations" linked across national and international networks. Particular attention was to be paid to how the new software would manage the caching, clustering and movement between storage media and across networks of collections of physics objects used in the analysis. In order to ensure that the project would immediately benefit the physics goals of CMS and US CMS while carrying out its technical R&D, the project also called for the use of the CACR computing and data storage systems to produce Terabyte samples of fully-simulated signal and background events (with a focus on intermediate-mass Higgs searches) to be stored in the database. The plan was agreed to by all parties by the Spring of 1997, and work officially began in June 1997.

Progress

Rapid and sustained progress has been achieved over the last two years: we have built prototype database, reconstruction, analysis and (Java3D) visualization systems. This has allowed us to test, validate and begin to develop the strategies and mechanisms that will make the implementation of massive distributed systems for data access and analysis in support of the LHC physics program possible. These systems will be dimensioned to accommodate the volume (measured in PetaBytes) and complexity of the data, the geographical spread of the institutes and the large numbers of physicists participating in each experiment. At the time of this writing the planned investigations of data storage and access methods, performance and scalability of the database, and the software development process, are all completed, or currently underway.

Current work includes the deployment and tests of the Terabyte-scale database at a few US universities and laboratories participating in the LHC program. In addition to providing a source of simulated events for evaluation of the design and discovery potential of the CMS experiment, the distributed database system will be used to explore and develop effective strategies for distributed data access and analysis at the LHC. These tests are foreseen to use local, regional (CalREN-2) and the Internet-2 backbones nationally, to explore how the distributed system will work, and which strategies are most effective.

The GIOD Project is due to complete at the end of 1999.

Computing Infrastructure

We have adopted several key technologies that will probably play significant roles in the LHC computing systems: OO software (C++ and Java), commercial OO database management systems (ODBMS; specifically Objectivity/DB), hierarchical storage management systems (HPSS) and fast networks (ATM LAN and OC12 regional links). The kernel of our prototype is a large (~1 Terabyte) Object database containing ~1,000,000 fully simulated LHC events. Using this database, we have investigated scalability and clustering issues in order to understand the performance of the database for physics analysis. Tests included making replicas of portions of the database, by moving objects in the WAN, executing analysis and reconstruction tasks on servers that are remote from the database, and exploring schemes for speeding up the selection of small sub-samples of events. Another series of tests involves hundreds of "client" processes simultaneously reading and/or writing to the database, in a manner similar to simultaneous use by hundreds of physicists, or in a data acquisition farm.

The GIOD project uses several of the latest hardware and software systems:

1. The Caltech HP Exemplar, a 256-PA8000 CPU SMP machine of ~3,000 SPECInt95

2. The High Performance Storage System (HPSS) from IBM

3. The Objectivity/DB Object Database Management System

4. An HP-C200 Workstation equipped with a dedicated 155 MBits/sec optical fiber link to the Exemplar (via a FORE ATM switch)

5. Several Pentium II-class PCs running Windows/NT, including an HP "Kayak" with a special "fx4" graphics card, and 256 MBytes RAM

6. Various high speed LAN and WAN links (ATM/Ethernet)

7. C++ and Java/Java3D

The Caltech Exemplar

The Exemplar is a NUMA machine with 64 GBytes of main memory, shared amongst all 256 processors. It runs the SPP-UX Unix operating system, which is binary compatible with HP-UX. The processors are interconnected using a CTI toroidal wiring system, which gives excellent low-latency internode communication. Two HiPPI switches also connect the nodes. There is over 1 TeraByte of disk attached to the system, which can achieve up to 1 GigaByte/sec parallel reads and writes. Fast Ethernet and ATM connections are available. The machine is located at, and operated by, the Caltech Centre for Advanced Computing Research (CACR). Funding for the machine is by a joint Caltech/JPL/NASA project, not related to GIOD.

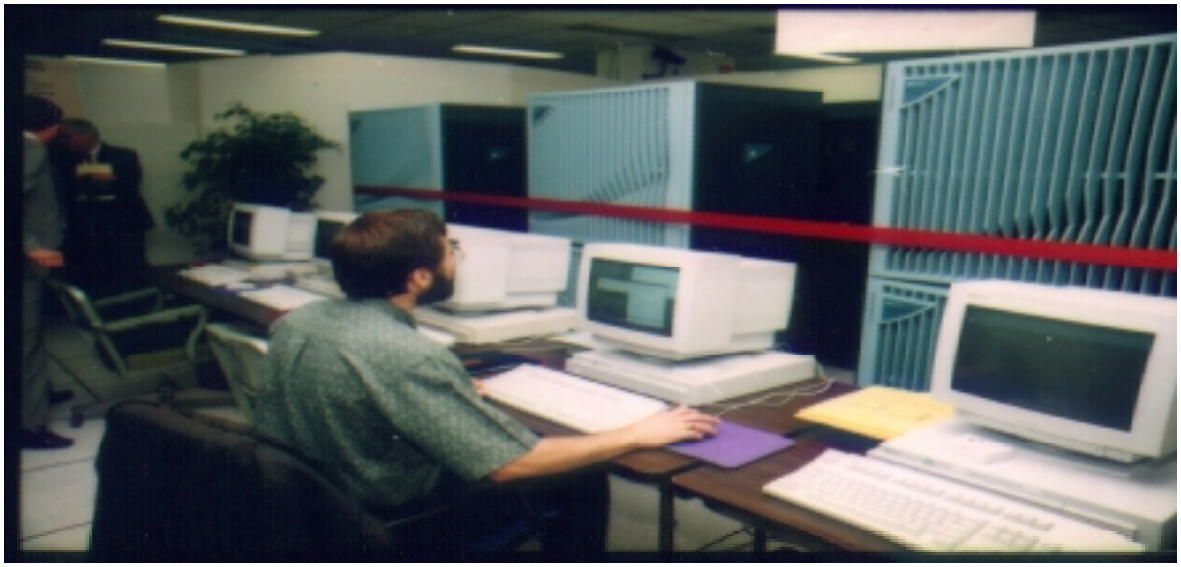


Figure 1: The 256 CPU Caltech Exemplar, the largest shared memory system in the world

We are also using several Pentium-class PCs running Windows/NT, and a couple of HP model 755 workstations. An HP model C200 workstation equipped with 155 Mbit/sec ATM has been installed, and this is directly fibre-attached (via a FORE ATM switch) to the Exemplar.

The Objectivity Database

The Objectivity/DB ODBMS is licensed for the Exemplar and workstations being used in the project. It is a pure object database that includes C++ and Java bindings, federations of individual databases, and the possibility of wide area database replication. We have also installed the Versant ODBMS, and have compared its functionality and inter-ODBMS migration issues.

The Objectivity/DB architecture comprises a “federation” of databases. All databases in the federation share a common Object scheme, and are indexed in a master catalog. Each database is a file. Each database contains one or more “containers”. Each container is structured as a set of “pages” (of a unique size) onto which the persistent objects are mapped. The database can be accessed by clients which are applications linked against the Objectivity/DB libraries and the database schema files. Client access to local databases is achieved via the local file system. Access to remote databases is made via an “Advanced Multithreaded Server” which returns database pages across the network to the client. Database locks are managed by a lockserver process. Locks operate at the database container level.

Networking

The Exemplar, HPSS and workstation systems are interconnected on the LAN using standard Ethernet and/or with HiPPI. Wide area connections between these systems and CERN (over a 4 Mbit/sec trans-Atlantic link) are being used in tests of distributed database "federations".

A "CENIC" OC12 link between CACR and the San Diego Supercomputer Centre (SDSC) will be used for WAN database tests between the Exemplar and a large peer system in San Diego. In addition, CACR will avail itself of Internet 2/NGI national connections and peering with ESNET in 1999.

Database scalability and access tests

We developed a simple scaling test application, and looked at the usability, efficiency and scalability of the Object Database while varying the number of objects, their location, the object selection mechanism and the database host platform.

The "Stars" Application

Overview of the application

The "stars" application is code that addresses the following problem domain: A region of the sky is digitized at two different wavelengths, yielding two sets of candidate bright objects, each characterised by a position (x,y) and width (sigma). The problem is to find bright objects at the same positions in both sets, consistent with the width of each. The schema used in the application specifies "star" objects with the data members xcentre,ycentre, sigma (width), catalogue (identification number) with associated member functions that return the position, the sigma, the catalogue number, the proximity of a point (X,Y) to the "star", and so on.

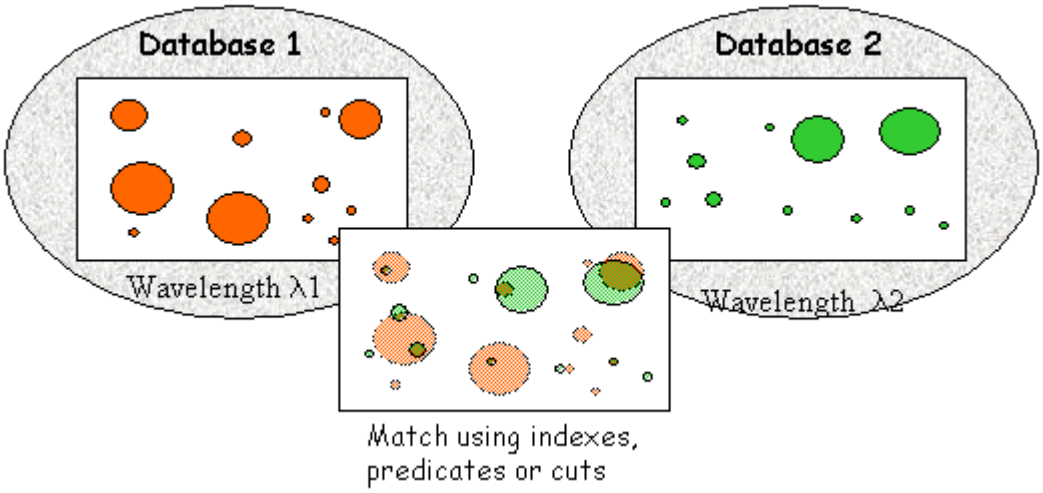


Figure 2: Showing the "Stars" application problem domain: matching bright objects from two scans of the sky at different wavelengths.

The application is in two parts: the first part generates a randomly-distributed set of stars in each of two databases in an Objectivity federated database. The second part attempts to match the positions of each star in the first database with each star in the second database, in order to find the star in the second database that is most close to the star in the first. We expect the matching time to scale as N2, where N is the number of stars in each database.

This application, although not taken from High Energy Physics, is analogous to matching energy deposits in a calorimeter with track impact positions, which is a typical event reconstruction task. The application has the advantage that it is small, and easy to port from one OS to another, and from one ODBMS to another

Details of the application

- Create a federated database with a star database for each wavelength
- Create a container in each database, and generate in each a number of stars with random positions and widths

- “loop” over all stars in one database, and for each one, find the nearest star in the other, using a proximity function
    - Proximity(x,y) = exp(- sqrt((x-xcentre)**2 + (y-ycentre)**2) / sigma)
    - where (x,y) is the position of the candidate, and xcentre, ycentre and sigma are the position and width of the target star.
- Using the class iterators provided by Objectivity, with a selection using standard C++ e.g.
    - if(xlow<x && x<xhigh && ylow<y && y<yhigh)
- Using the iterator with an Objectivity predicate e.g.
    - sprintf(pred,''x>%f && x<%f && y>%f && y<%f'',xlow,xhigh, ylow,yhigh)
    - Itr.scan(dbH,oocRead,oocAll,pred)
- Using an index on (x,y) in the other database, with a lookup key ooGreaterThanLookupField

Results from the Scalability Tests using “Stars”

Using the "stars" application, we measured matching speed as a function of the number of objects in each database. The results showed that the fastest matching times are obtained by using an index on the positions of the star objects, and the slowest times with a text-based predicate" selection.

We then measured matching speeds on different hardware and operating systems, comparing the performance of the matching using indices on the Exemplar, a Pentium II (266 MHz), a Pentium Pro (200 MHz) and an HP 755 workstation. For all these tests, we ensured that both stars databases were completely contained in the system cache, so that we could disregard effects due to disk access speeds on the various machines. The results demonstrated the platform independence of the ODBMS and application software, and illustrates the performance differences due to the speeds of the CPUs, the code generated by the C++ compilers, and so on.

Another test showed how the application and database could reside on different systems, and what impact on performance there was if they did: we measured the matching speed on a 266 MHz Pentium II PC with local databases, and with databases stored on a remote HP 755 workstation. For the problem sizes we were using, there was no significant performance degradation when the data were held remotely from the client application.

The Objectivity/DB cache is used to store one or more pages of the database(s) in memory, so improving performance for queries that access objects contained in cached pages. The size of the cache can be configured in the application code. We measured the behaviour of the "stars" application performance with differing cache sizes when matching 2000 star objects. In this test we observed some erratic behaviour when using very small caches. However, the overall results showed that, for the databases being used, the objects were all accomodated in the default sized cache, and no benefit was obtained by increasing it.

Finally, we measured the speed at which large numbers of databases could be created within a single Objectivity federation. The results are shown below:

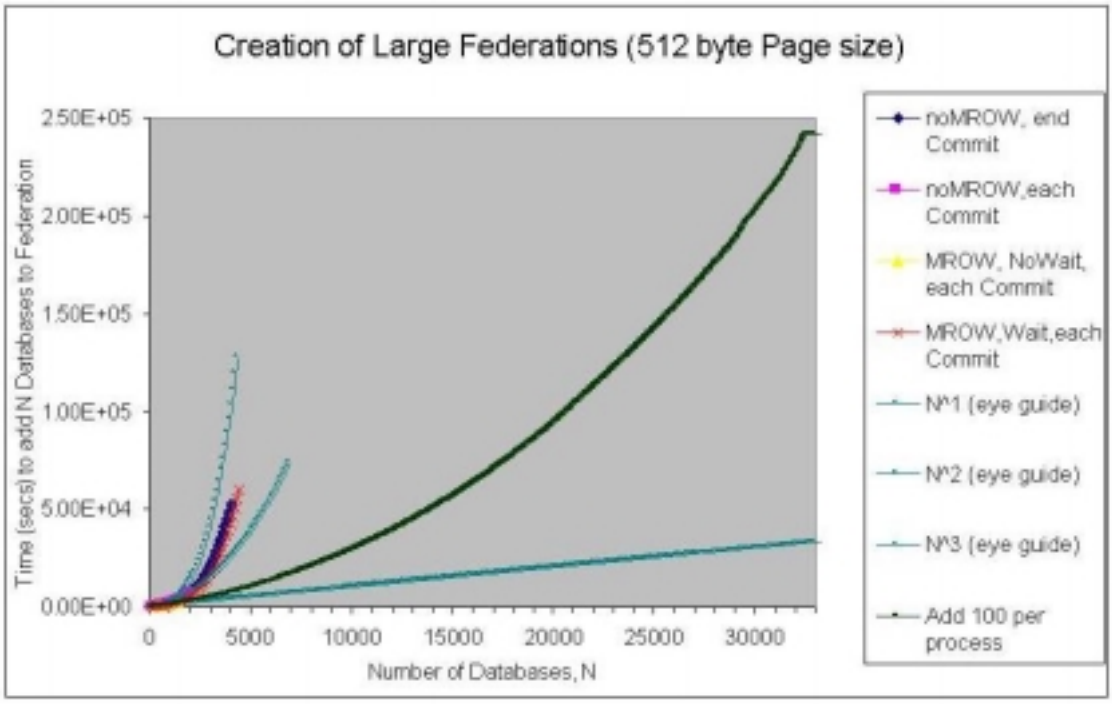


Figure 3: Showing the tim taken to add new databases to an existing Objectivity federation. We were able to create a federation of 32,000 databases before boredom set in.

Summary of Scalability Tests with "Stars"

Our results demonstrated the platform independence of both the database and the application, and the locality independence of the application. We found, as expected, significance query performance gains when objects in the database were indexed appropriately.

Scalability tests on the Caltech Exemplar

The scalability tests were performed on the HP Exemplar machine at Caltech. As described already, this is a 256 CPU SMP machine of some 0.1 TIPS. There are 16 nodes, which are connected by a special-purpose fast network called a CTI. Each node contains 16 PA8000 processors and one node file system. A node file system consists of 4 disks with 4-way striping, with a file system block size of 64 KB and a maximum raw I/O rate of 22 MBytes/second. We used up to 240 processors and up to 15 node file systems in our tests. We ensured that data was always read from disk, and never from the file system cache. An analysis of the raw I/O behaviour of the Exemplar can be found in [5].

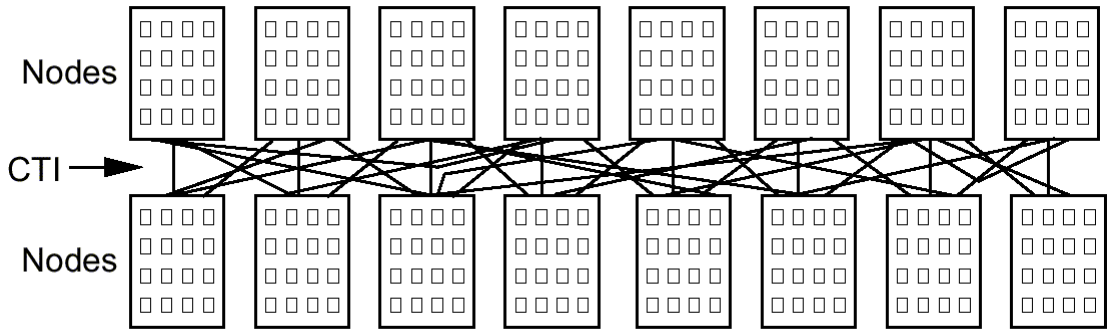


Figure 4: Configuration of the Hp Exemplar at Caltech

The Exemplar runs a single operating system image, and all node file systems are visible as local UNIX file systems to any process running on any node. If the process and file system are on different nodes, data is transported over the CTI. The CTI was never a bottleneck in the test loads we put on the machine: it was designed to support shared memory programming and can easily achieve data rates in the GByte/second range. As such, the Exemplar can be thought of as a farm of sixteen 16-processor UNIX machines with cross-mounted file systems, and a semi-infinite capacity network. Though the Exemplar is not a good model for current UNIX or PC farms, where network capacity is a major constraining factor, it is perhaps a good model for future farms which use GByte/second networks like Myrinet as an interconnect.

The object database tested was the HP-UX version of Objectivity/DB.

We made two sets of tests, completed with different database configurations and different data.

Tests with synthetic data

Our first round of tests used synthetic event data represented as sets of 10 KByte objects. A 1 MByte event thus became a set of 100 objects of 10 KB. Though not realistic in terms of physics, this approach does have the advantage of giving cleaner results by eliminating some potential sources of complexity.

For these tests we used Objectivity/DB v4.0.10. We placed all database elements (database clients, database lockserver, federation catalog file) on the Exemplar itself. Database clients communicated with the lockserver via TCP/IP sockets, but all traffic was local inside the supercomputer. The federation catalog and the payload data were accessed by the clients though the Exemplar UNIX filesystem interface.

The test loads were generated with the TOPS framework which runs on top of Objectivity.

Two things in the Objectivity architecture were of particular concern. First, Objectivity does not support a database page size of 64 KB, it only supports sizes up to 64 KB minus a few bytes. Thus, it does not match well to the node file systems which have a block size of exactly 64 KB. After some experiments we found that a database page size of 32 KB was the best compromise, so we used that throughout our tests. Second, the Objectivity architecture uses a single lockserver process to handle all locking operations. This lockserver could become a bottleneck when the number of (lock requests from) clients increases.

Reconstruction test

We have tested the database under an event reconstruction workload with up to 240 clients. In this workload, each client runs a simulated reconstruction job on its own set of events. For one event, the actions are as follows:

- **Reading**: 1 MB of 'raw' data is read, as 100 objects of 10 KB. The objects are read from 3 containers: 50 from the first, 25 from the second, and 25 from the third. Inside the containers, the objects are clustered sequentially in the reading order.
- **Writing**: 100 KB of 'reconstructed' data is written, as 10 objects of 10KB, to one container.
- **Computation:** 2.10^3 MIPSs are spent per event (equivalent to 5 CPU seconds on one Exemplar CPU).

Reading, writing, and computing are interleaved with one another. The data sizes are derived from the CMS computing technical proposal. The proposal predicts a computation time of 2.10^4 MIPSs per event. However, it also predicts that CPUs will be 100 times more powerful (in MIPS per $) at LHC startup in 2005. We expect that disks will only be a factor 4 more powerful (in MBytes/second per $) in 2005. In our test we chose a computation time of 2.10^3 MIPSs per event as a compromise. The clustering strategy for the raw data is based on work described in [9]. The detector is divided into three separate parts and data from each part are clustered separately in different containers. This allows faster access for analysis tasks which only require some parts of the detector. The database files are divided over four Exemplar node file systems, with the federation catalog and the journal files on a fifth file system. In reading the raw data, we used the read-ahead optimisation described in a later section.

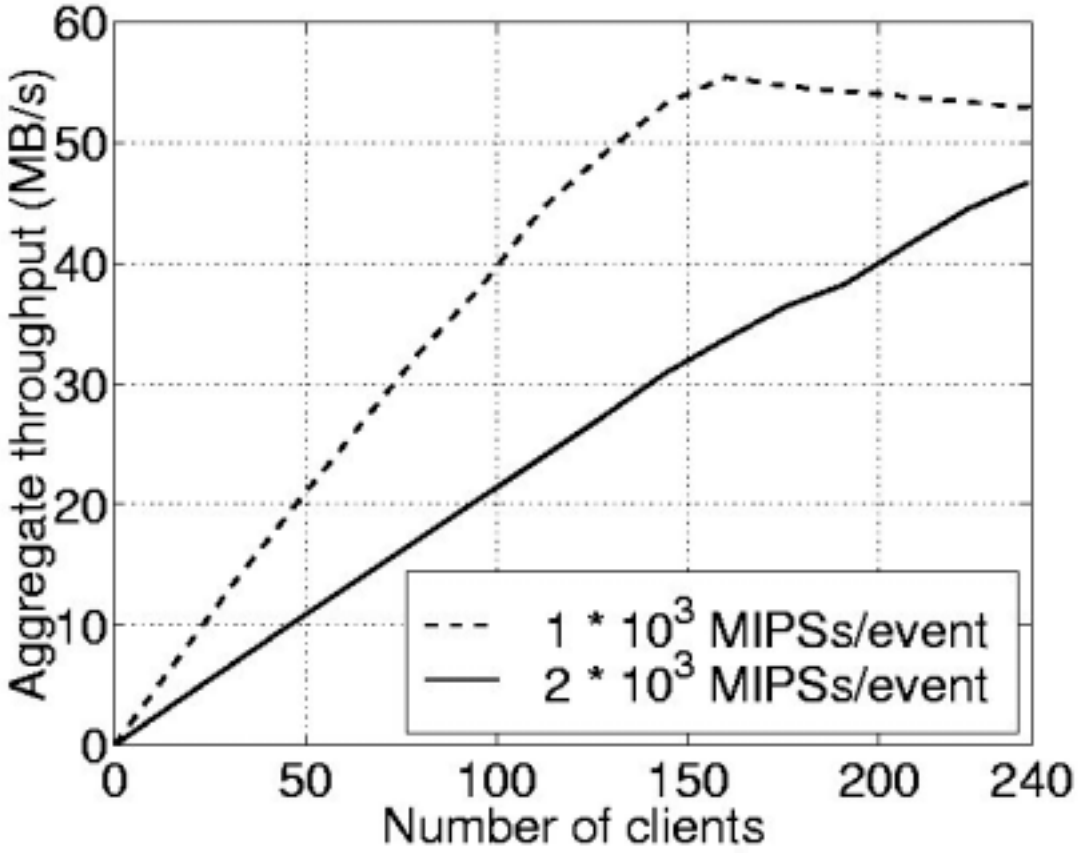


Figure 5: Scalability of Reconstruction workloads

The results from our tests are shown in the above Figure. The solid curve shows the aggregate throughput for the CMS reconstruction workload described above. The aggregate throughput (and thus the number of events reconstructed per second) scales almost linearly with the number of clients. In the left part of the curve, 91% of the allocated CPU resources are spent running actual reconstruction code. With 240 clients, 83% of the allocated CPU power (240 CPUs) is used for physics code, yielding an aggregate throughput of 47 MBytes/second (42 events/s), using about 0.1 TIPS.

The dashed curve shows a workload with the same I/O profile as described above, but half as much computation. This curve shows a clear shift a from CPU-bound to a disk-bound workload at 160 clients. The maximum throughput is 55 MBytes/second, which is 63% of the maximum raw throughput of the four allocated node file systems (88 MBytes/second). Overall, the disk efficiency is less good than the CPU efficiency. The mismatch between database and file system page sizes discussed above is one obvious contributing factor to this. In tests with fewer clients on a platform with a 16 KByte file system page size, we have seen higher disk efficiencies for similar workloads.

The read-ahead optimization

When reading raw data from the containers in the above reconstruction tests, we used a read-ahead optimisation layer built into our testbed. The layer takes the form of a specialised iterator, which causes the database to read containers in bursts of 4 MByte (128 pages) at a time. Without this layer, the (simulated) physics application would produce single page reads interspersed with computation. Tests have shown that such less bursty reading leads to a loss of I/O performance.

We consider the case of *N* clients all iterating through *N* containers, with each client accessing one container only. The computation in each client is 2.10^3 MIPSs per MB read. Containers are placed in databases on two node file systems, which have a combined raw throughput of 44 MBytes/second.

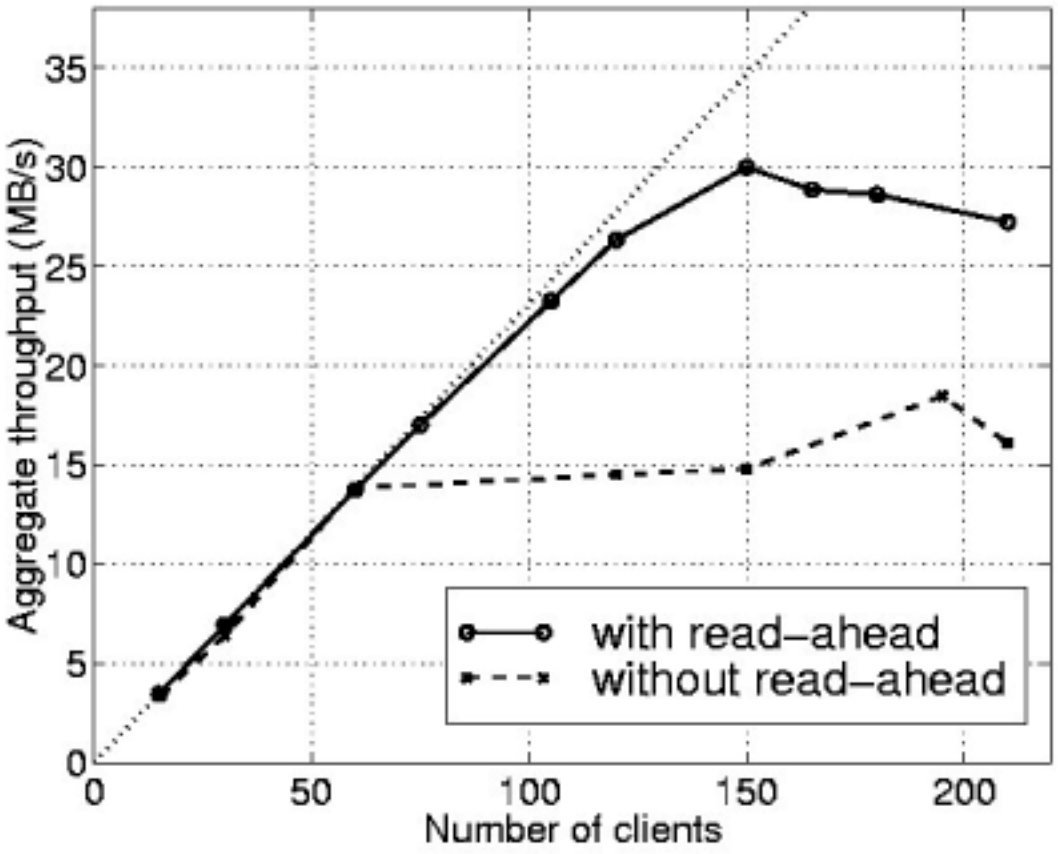


Figure 3: Performance of many clients all performing sequential reading on a container

Figure 3 shows that without the read-ahead optimisation, the workload becomes disk-bound fairly quickly, at 64 clients. Apparently, a lot of time is lost in disk seeks between the different containers. In this test, the lack of a read-ahead optimisation degrades the maximum I/O performance with a factor of two. Because of the results in [9], we expect that the performance would have been degraded even more in the reconstruction test of section 3, where each client reads from three containers.

DAQ (Data Acquisition) test

We do not currently advocate the use of an object database as the primary storage method in a real-time DAQ system. We feel that currently, the most attractive approach still is to stream data to flat files, and to then convert these files into objects in quasi-realtime. We have tested the database with such a quasi-realtime data acquisition workload up to 238 clients.

In this test, each client is writing a stream of 10 KByte objects to its own container. For every event (1 MByte raw data) written, about 180 MIPSs (0.45 CPU seconds on the Exemplar) are spent in simulated data formatting. For comparison, 0.20 CPU seconds are spent by Objectivity in object creation and writing, and the operating system spends 0.01 CPU seconds per event. No read operations on flat files or network reads are done by the clients. The database files are divided over eight node file systems, with the federation catalog and the journal files on a ninth file system.

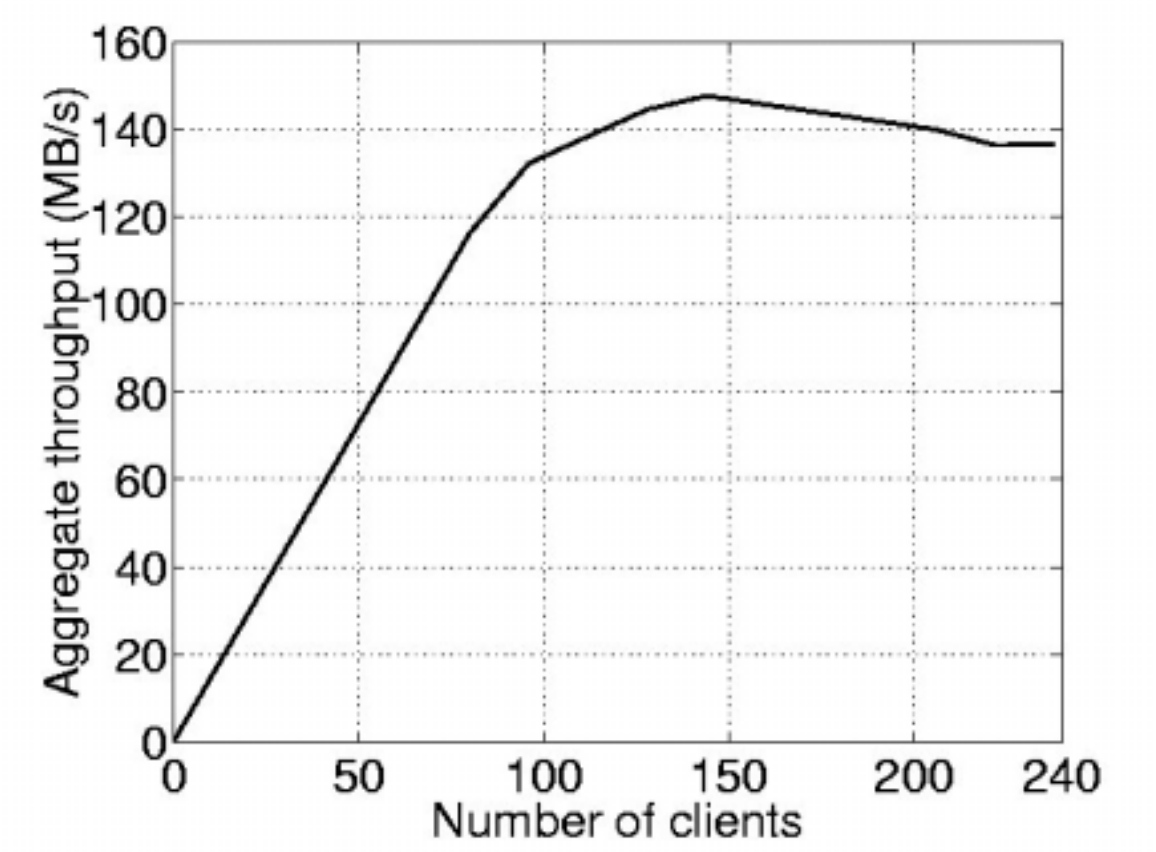


Figure 6: Scalability of a DAQ workload

The test results are shown in the above Figure. Again we see a transition from a CPU-bound to a disk-bound workload. The highest throughput is 145 MBytes/second at 144 clients, which is 82% of the maximum raw throughput of the eight allocated node file systems (176 MBytes/second).

In workloads above 100 clients, when the node file systems become saturated with write requests, these file systems show some surprising behaviour. It can take very long, several minutes, to perform basic operations like syncing a file (which is done by the database when committing a transaction) or creating a new (database) file. We believe this is due to the appearance of long 'file system write request' queues in the operating system. During the test, other file systems not saturated with write requests still behave as usual. We conclude from this that one should be careful in saturating file systems with write requests: unexpected long slowdowns may occur.

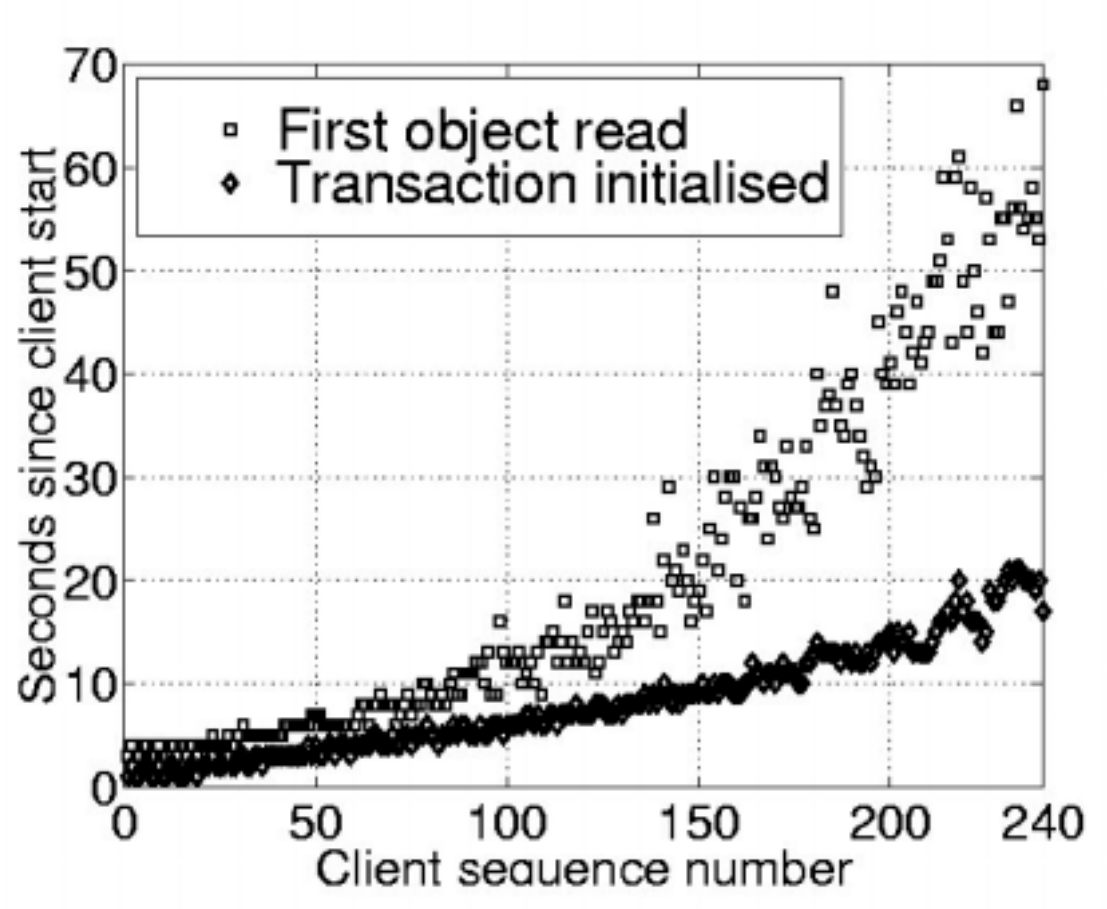


Figure 7: Client startup in the 1.103 MIPSs reconstruction test

Client startup

We measured the scalability of client startup times throughout our tests. We found that the client startup time depends on the number of clients already running and on the number of clients being started at the same time. It depends much less on the database workload, at least if the federation catalog and journal files are placed on a file system that is not heavily loaded. With heavily loaded catalog and journal file systems, startup times of many minutes have been observed.

The above Figure shows a startup time profile typical for our test workloads. Here, new clients are started in batches of 16. For client number 240, the time needed to open the database and initialise the first database transaction is about 20 seconds. The client then opens four containers (located in three different database files), reads some indexing data structures, and initialises its reconstruction loop. Some 60 seconds after startup, the first raw data object is read. If a single new client number 241 is started by itself, opening the database and initialising the transaction takes some 5 seconds.

Tests with real physics data

Our second round of tests wrote realistic physics event data into the database. These data were generated from a pool of around one million fully simulated LHC multi-jet QCD events. The simulated events were used to populate the Objectivity database according to an object scheme

that fully implemented the complex relationships between the components of the events. These events, and how they were generated and reconstructed, is more fully described below.

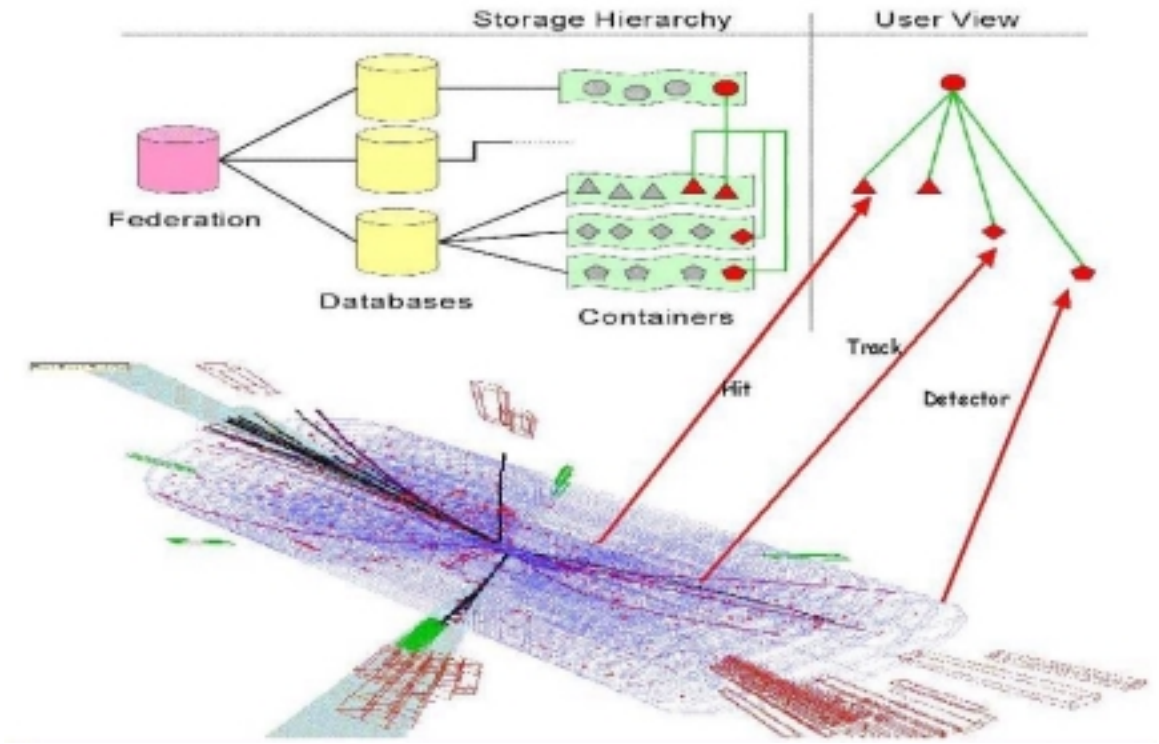


Figure 8: The mapping of the event objects to the Object Database components.

In these tests we used Objectivity/DB v5.0. Only the database clients and the payload databases were located on the Exemplar system. The lockserver was run on an HP workstation connected to the Exemplar via a LAN. The database clients contacted the lockserver over TCP/IP connections. The federation catalog was placed on a C200 HP workstation, connected to the Exemplar over a dedicated ATM link (155 Mbits/second). The clients accessed the catalog over TCP/IP connections to the Objectivity/DB AMS server, which ran on the C200 workstation.

Each database client first read 12 events into memory, then wrote them out repeatedly into its own dedicated database file. Once the database file reached a size of about 600 MBytes, it was closed and deleted by the client. Then the client created and filled a new database file. This was arranged to avoid exhausting file system space during the tests. In a real DAQ system, periodic switches to new database files would also occur, whilst retaining the old database files.

Database files were evenly distributed over 15 node file systems on the Exemplar. Of these node file systems, ten are rated at 22 Mbytes/second raw, the remaining five contain fewer disks and achieve a lower throughput.

Results from tests with real physics data

We present the results from two measurements. In the first measurement, event data was written into a single container in the client database file. In the second measurement, the six objects making up each event were written to separate containers in the same database file. Associations between the objects in the event were created.

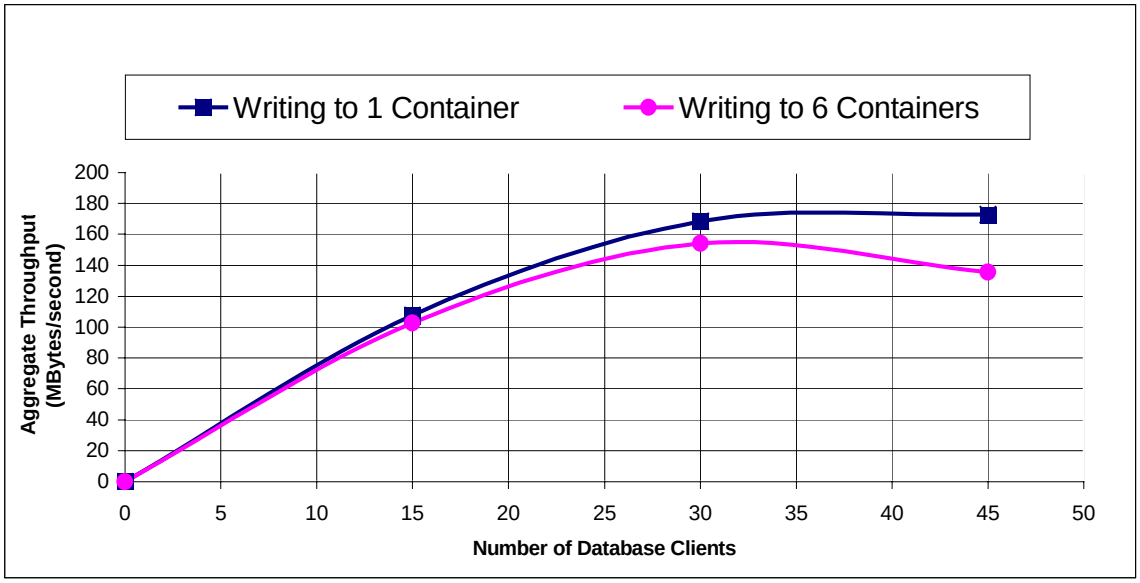


Figure 5: DAQ tests with real physics data

Figure 5 shows the test results. The first measurement shows a best rate of 172 MBytes/second, reached with 45 running clients. In the second measurement a rate of 154 MBytes/second was achieved when running with 30 clients. We note that the overhead associated with writing into six separate containers is not significant.

In related tests we ran more than 45 clients, but observed that only about 40 to 60 clients were actively writing to the database at the same time. All remaining clients were busy closing their database file (when the 600 MByte size limit was reached), then deleting it and creating a new one. This effect is not currently understood.

Comments on the Objectivity lockserver

The lockserver, whether run remotely or locally on the Exemplar, was not a bottleneck in any of our tests. From a study of lockserver behaviour under artificial database workloads with a high rate of locking, we estimate that lockserver communication may become a bottleneck in a DAQ scenario above 1000 MB/s.

Conclusions from the Exemplar scalability tests

In the first series of tests, with all components of the Objectivity/DB system located on the Exemplar, we observed almost ideal scalability, up to 240 clients, under synthetic physics reconstruction and DAQ workloads. The utilisation of allocated CPU resources on the Exemplar is excellent, with reasonable to good utilisation of allocated disk resources. It should be noted that the Exemplar has a very fast internal network.

In the second series of tests the database clients were located on the Exemplar, and the Objectivity lockserver, AMS and catalog were located remotely. In this configuration, the system achieved aggregate write rates into the database of more than 170 MBytes/second. This exceeds the 100 MBytes/second required by the DAQ systems of the two main LHC experiments.

The Versant ODBMS

We evaluated the usability and performance of Versant ODBMS, Objectivity's main competitor. Based on these tests we concluded that Versant would offer an acceptable alternative solution to Objectivity, if required.

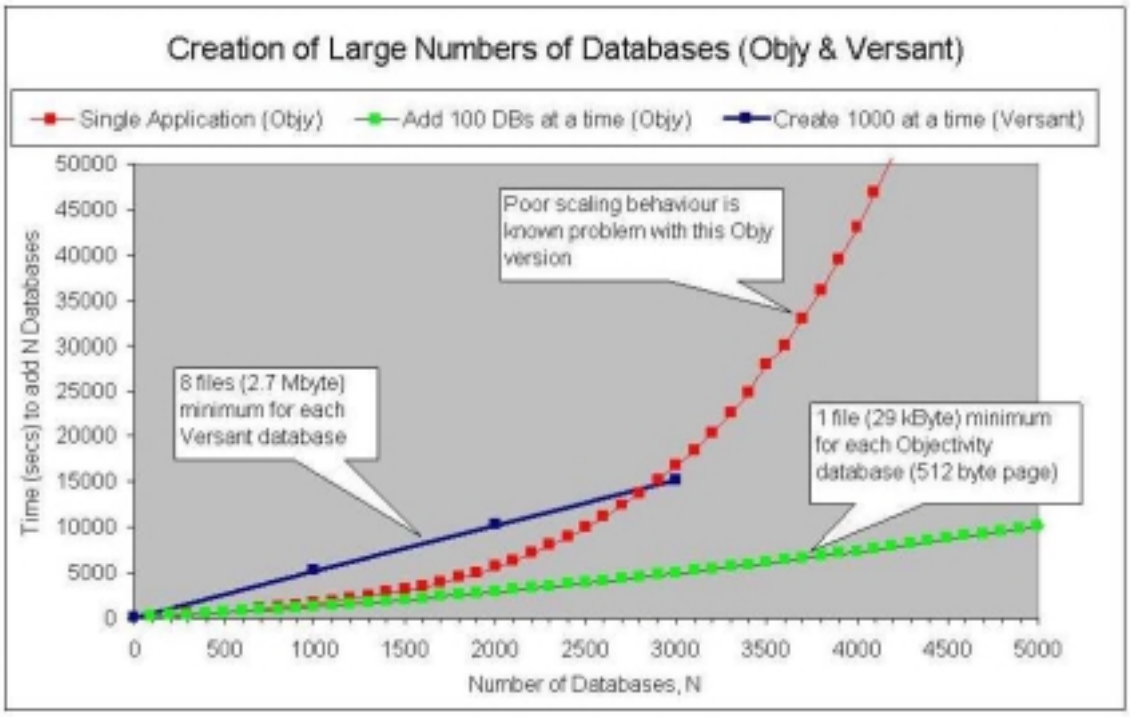


Figure 9: Time to create databases as a function of the number already existing: Objectivity and Versant

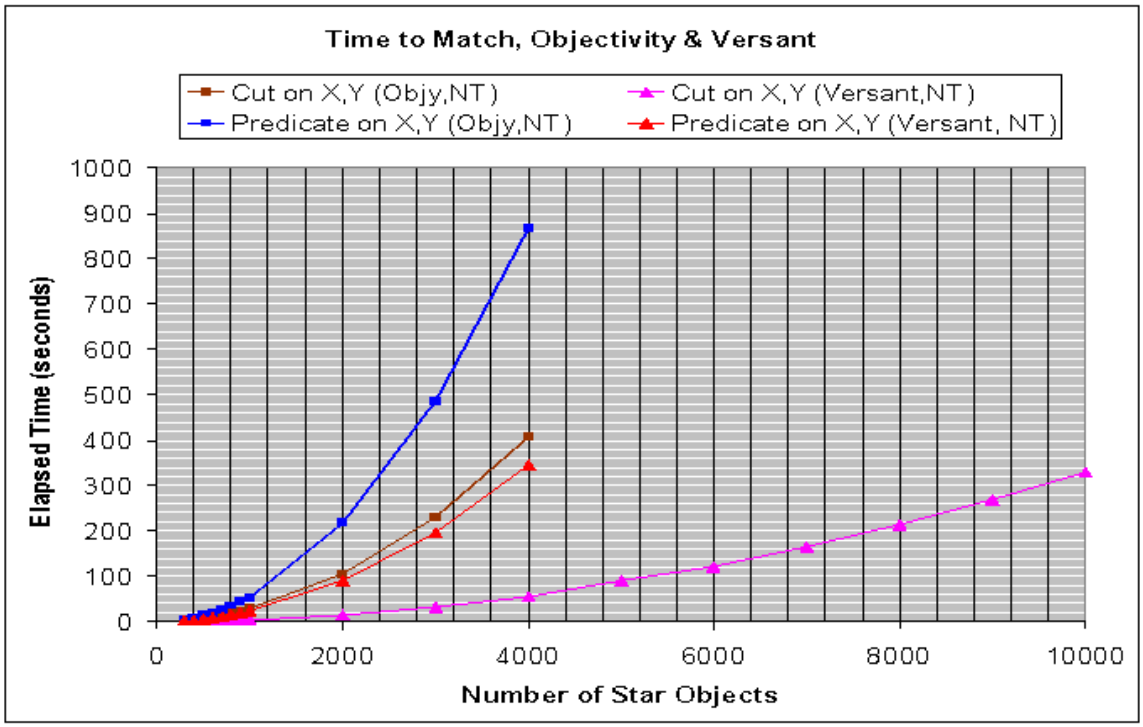


Figure 10: Time to match objects using the "stars" application: Objectivity and Versant

Code comparisons: Objectivity and Versant

We spent some time converting our “stars” application (see above) from Objectivity to Versant. This process took a couple of days. In the following tables, we show differences between the APIs and the mechanics of creating and populating databases.

Table 1: Objectivity and Versant database creation semantics

Objectivity	Versant
Create new federated database oonewfd -lockserver pcbunn -fdfilepath f:\stars\stars.fdb starsdb Imbue it with the stars “schema” ooddlx stars.ddl starsdb Compile the source file cl.exe … stars.cpp Compile the file created by “ooddlx” cl.exe … stars_ddl.cpp Link the application link.exe … stars.obj stars_ddl.obj …	Make new group database placeholders makedb -g starsdb1 makedb -g starsdb2 Create the group database files createdb starsdb1 createdb starsdb2 Create the “schema” .sch from the .imp file schcomp -I. -I\versant\5_0_7\NT\h schema.imp Imbue the databases with the schema sch2db -D starsdb1 -y schema.sch sch2db -D starsdb2 -y schema.sch Compile the application and methods files cl.exe … stars.cpp cl.exe … methods.cpp Compile the file created by “schcomp” cl.exe … schema.cxx Link the application link.exe … stars.obj methods.obj schema.obj ...

Table 2: Comparing the use of Iterators and Lists between Objectivity and Versant

Objectivity	Versant

cHandle1.open(dbHandle1,"DB1_CONT",oocRead);
cHandle2.open(dbHandle2,"DB2_CONT",oocRead);
 	
if(star_iterator1.scan(cHandle1,oocRead)) {
time(&start);
while (star_iterator1.next()) {
// find 3-sigma value for this star ...
sigma3 = 3.0*star_iterator1->get_sigma();
xlow = star_iterator1->get_x() - sigma3;
xhigh = xlow + 2.*sigma3;
ylow = star_iterator1->get_y() - sigma3;
yhigh = ylow + 2.*sigma3;
nearest = -1;
 	
if(star_iterator2.scan(cHandle2,oocRead)) {
dmax = 0.0;
while (star_iterator2.next()) {
x = star_iterator2->get_x();
y = star_iterator2->get_y();
if(x>xlow && x<xhigh && y>ylow && y<yhigh) {
d = star_iterator1->proximity(x,y);
if (d>dmax) {
dmax = d;
nearest = star_iterator2->get_catalogue();
}
}
}
checksum += ~nearest;
}
time(&finish);
cout << "Checksum " << checksum << endl;
}
	LinkVstr<Star> a = Star::Find_all();
	
	::dom->set_default_db("starsdb2");
	LinkVstr<Star> b = Star::Find_all();
	
	for(i=0;i<a.size();i++) {
	// find 3-sigma value for this star ...
	sigma3 = 3.0*a[i]->get_sigma();
	xlow = a[i]->get_x() - sigma3;
	xhigh = xlow + 2.*sigma3;
	ylow = a[i]->get_y() - sigma3;
	yhigh = ylow + 2.*sigma3;
	dmax = 0.0;
	nearest = -1;
	for(j=0;j<b.size();j++) {
	x = b[j]->get_x();
	y = b[j]->get_y();
	if(x>xlow && x<xhigh && y>ylow && y<yhigh) {
	d = a[i]->proximity(x,y);
	if(d>dmax) {
	dmax = d;
	nearest = b[j]->get_catalogue();
	}
	}
	}
	checksum += ~nearest;
	}
	b.release();
	a.release();
	time(&finish);
	cout << "Checksum " << checksum << endl;

Table 3: Comparing database population

Objectivity	Versant

// Get a handle on the container in the first database
 	
ooHandle(Collection) db1Cont = new("DB1_CONT",1,NPAGES,0,dbHandle1) Collection("Stars1");
 	
// and a handle on the container in the second
 	
ooHandle(Collection) db2Cont = new("DB2_CONT",1,NPAGES,0,dbHandle2) Collection("Stars2");
 	
cout << "Generating " << NSTARS << " stars " << endl;
 	
time(&start);
	// Generate NSTARS stars in the first database,
	
	cout << "Generating " << NSTARS << " stars " << endl;
	
	time(&start);
	
	for (i=0;i<NSTARS;i++) {
	x = drand48();
	y = drand48();
	sigma = 0.1*drand48();
	Star *star = O_NEW_PERSISTENT(Star)(x,y,sigma,i);
	}
	
	// change to the second database

```
for (i=0;i<NSTARS;i++) {
       x = drand48();
       y = drand48();
       sigma = 0.1*drand48();
       star          =          new(db1Cont)
Star(x,y,sigma,i);
     x = drand48();
       y = drand48();
       sigma = 0.1*drand48();
       star          =          new(db2Cont)
Star(x,y,sigma,i);
}

time(&finish);
```

```
::dom->set_default_db("starsdb2");

for (i=0;i<NSTARS;i++) {
       x = drand48();
       y = drand48();
       sigma = 0.1*drand48();
       Star  *star  =  O_NEW_PERSISTENT(Star)
(x,y,sigma,i);
}

time(&finish);
```

Table 4: Comparing the use of predicate searches

Objectivity	Versant

`Char pred[130];`
`sigma3 = 3.0*star_iterator1->get_sigma();`
`xlow = star_iterator1->get_x() - sigma3;`
`xhigh = xlow + 2.*sigma3;`
`ylow = star_iterator1->get_y() - sigma3;`
`yhigh = ylow + 2.*sigma3;`
 	
`(void)  sprintf(pred,"xcentre>%f  &&`
`xcentre<%f  &&  ycentre>%f  &&`
`ycentre<%f",xlow,xhigh,ylow,yhigh);`
 	
`if(star_iterator2.scan(cHandle2,oocRead,ooc`
`All,pred)) {`
`  dmax = 0.0;`
`  nearest = -1;`
`  while  (star_iterator2.next())  {`
 	
`    x = star_iterator2->get_x();`
`    y = star_iterator2->get_y();`
`    d  =  star_iterator1-`
`>proximity(x,y);`
`    if (d>dmax) {`
`      dmax = d;`
`      nearest  =  star_iterator2-`
`>get_catalogue();`
`    }`
`  }`
`}`
	`PPredicate pred;`
	
	`sigma3 = 3.0*a[i]->get_sigma();`
	`xlow = a[i]->get_x() - sigma3;`
	`xhigh = xlow + 2.*sigma3;`
	`ylow = a[i]->get_y() - sigma3;`
	`yhigh = ylow + 2.*sigma3;`
	
	`pred  =  PAttribute("Star::xcentre")  >  xlow`
	`&&`
	`  PAttribute("Star::xcentre")  <  xhigh`
	`&&`
	`  PAttribute("Star::ycentre") > ylow`
	`&&`
	`  PAttribute("Star::ycentre")  <`
	`yhigh;`
	
	`LinkVstr<Star> b =`
	`PClassObject<Star>::Object().select("starsd`
	`b2",FALSE,pred);`
	
	`dmax = 0.0;`
	`nearest = -1;`
	`for(j=0;j<b.size();j++) {`
	`  x = b[j]->get_x();`
	`  y = b[j]->get_y();`
	`  d = a[i]->proximity(x,y);`
	`  if(d>dmax) {`
	`    dmax = d;`
	`    nearest = b[j]->get_catalogue();`
	`  }`
	`}`

Object Database Replication from CERN to Caltech

We tested one aspect of the feasibility of wide area network (WAN)-based physics analysis by measuring replication performance between a database at CERN and one at Caltech. For these tests, an Objectivity/DB "Autonomous Partition" was created on a 2 GByte NTFS disk on one of the Pentium PCs at Caltech. This AP contained a replica of a database at CERN. At the same time, an AP was created at CERN with a replica of the same database. Then, an update of 2 kBytes was made every ten minutes to the database at CERN, so causing the replicas to be updated. The transaction times for the local and remote replications were measured over the course of one day.

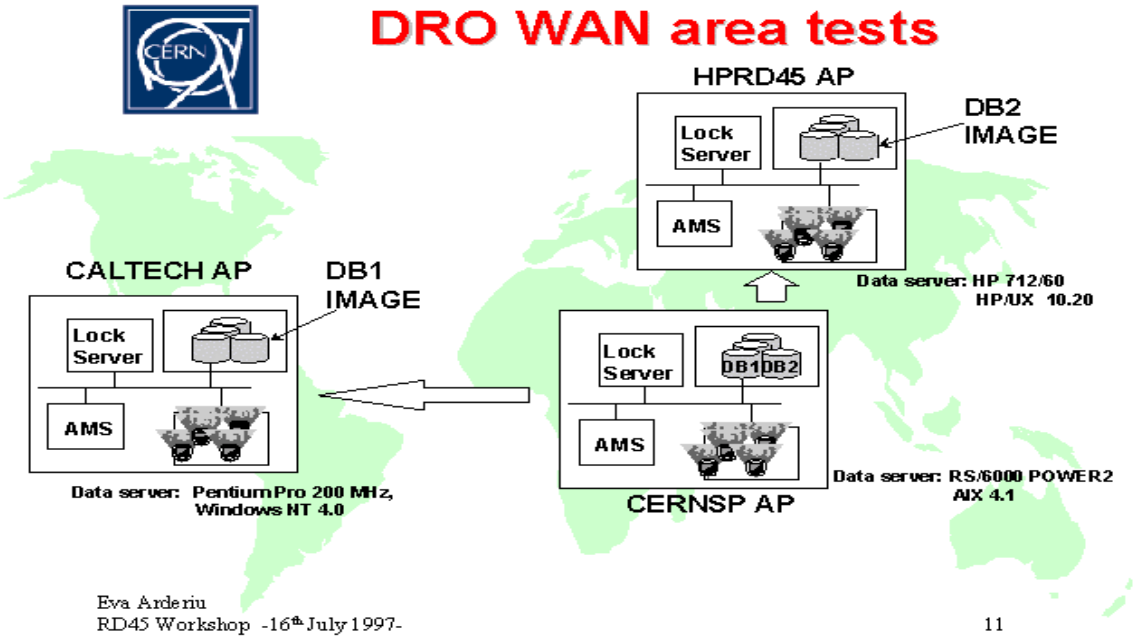


Figure 11: The DataReplicationOption wide area network test setup

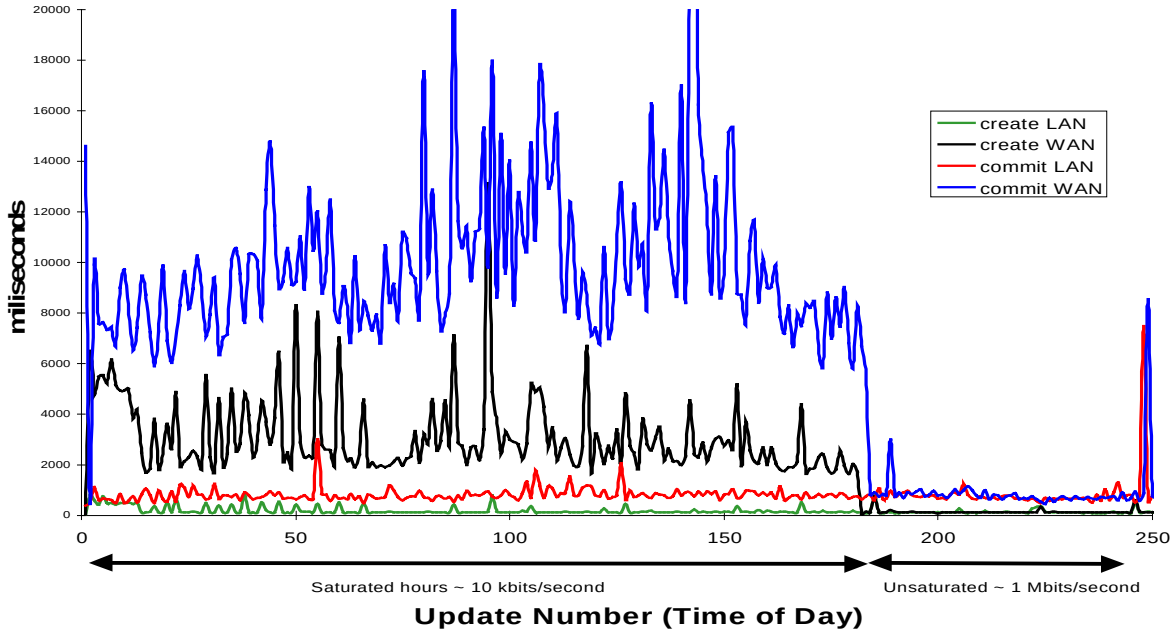


Figure 12: Time to achieve an update and commit it, as a function of time of day

The results show that during "saturated hours", when the WAN is busy, the time to commit the remote transaction is predictably longer than the time to commit the local transaction. On the other hand, when the WAN is quieter, the remote transaction takes no longer than the local transaction. This result demonstrates that, given enough bandwidth, databases may be transparently (and seamlessly) replicated from CERN to remote institutions.

The HPSS NFS Interface

We tested the operation of Objectivity/DB with a federated database located on an HPSS-managed NFS mounted file system. The HPSS machine at CACR exported a filesystem to an HP 755 workstation, where an Objectivity/DB installation was used to create a federation consisting of two 0.5 MBytes "stars" databases (see the description of the "stars" application above) located on the mounted filesystem. The matching application was run successfully. Then the database bitfiles were forced off HPSS disk and into tape, and the application again run. This caused an RPC timeout in the Objectivity application during the restore of the databases from tape to disk. We then inserted a call to "ooRpcTimeout" in the application, specifying a longer wait time, and re-ran the application successfully.

In addition, we tested the performance of the NFS-mounted HPSS filesystem for simple file copies. We copied a 300 MByte file from local disk on the HP 755 workstation into the NFS filesystem, and achieved a data transfer rate of ~330 kBytes/sec. This results shows that the system is reliable for large files, the data transfer rate approximating the available LAN bandwidth between the HP 755 and the HPSS server machine.

The CMS H2 Test Beam OO Prototype

We ported the data reconstruction and analysis software for the H2 detector test beam at CERN to Windows/NT. This task involved acquiring Rogue Wave Tools.h++ and installing an Objectivity/DB database of approximately 500 MBytes of raw data from CERN, previously copied across the WAN to Caltech. After initial tests with the software, it was decided to redirect all activities towards the CMSOO prototype described later.

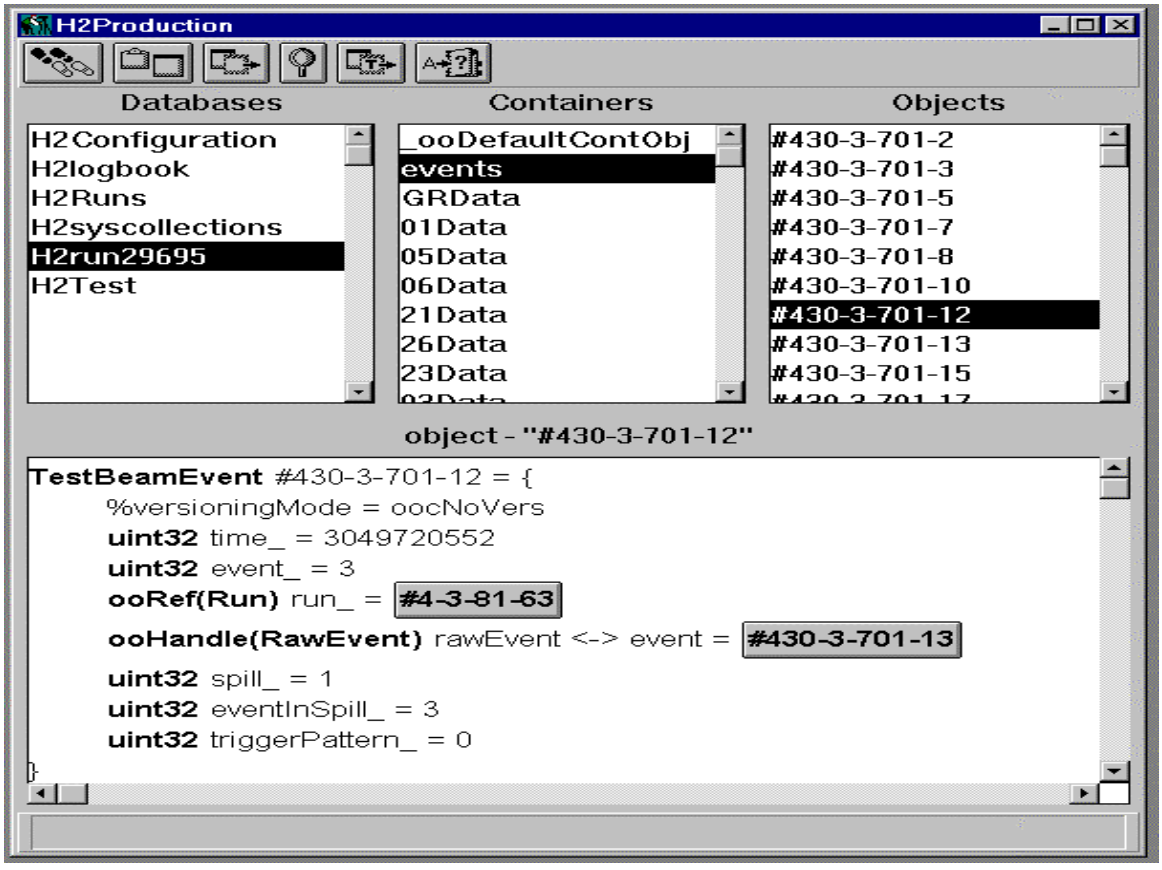


Figure 13: Showing the layout of the H2 Test Beam database, as installed on Windows NT at Caltech

CMSOO - Tracker, ECAL and HCAL software prototype

In 1998, CMS Physicists had produced several sub-detector orientated OO prototypes (e.g. for the Tracker, ECAL and HCAL detectors). These codes were mainly written in C++, occasionally with some Fortran, but without persistent objects. We took these codes and integrated them into an overall structure, redesigning and restructuring them where necessary. We then added persistency to the relevant classes, using the Objectivity/DB API. We then reviewed the code and its structure for speed, performance and effectiveness of the algorithms, and added global reconstruction aspects. These included track/ECAL cluster matching, jet finding and event tagging.

Concurrently, Caltech/HEP submitted a successful proposal to NPACI (The National Partnership for Advanced Computing Infrastructure) that asked for an Exemplar allocation to generate ~1,000,000 fully-simulated multi-jet QCD events. Event simulation has progressed since then, to an accumulated total of ~1 TBytes of data (~1,000,000 events), stored in HPSS. The events were used as a copious source of "raw" LHC data for processing by the "CMSOO" application and population of the GIOD database.

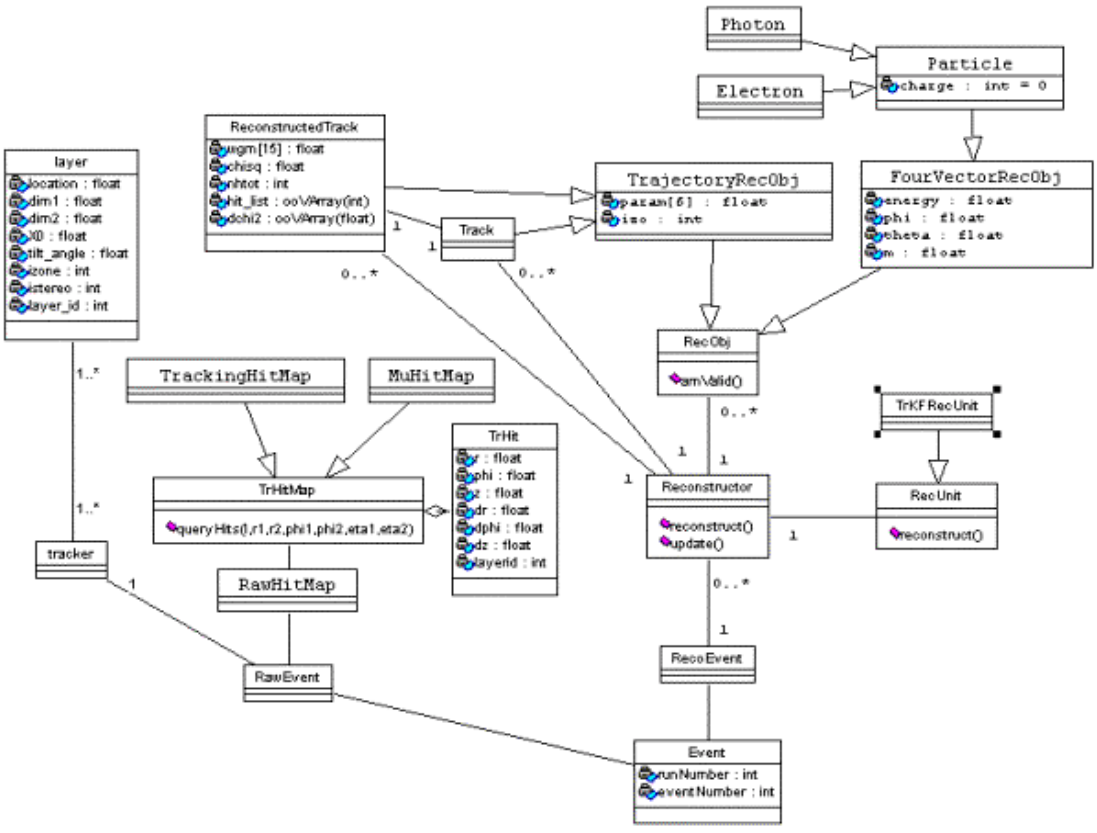


Figure 14: The class schema diagram for the track and calorimeter CMSOO prototype

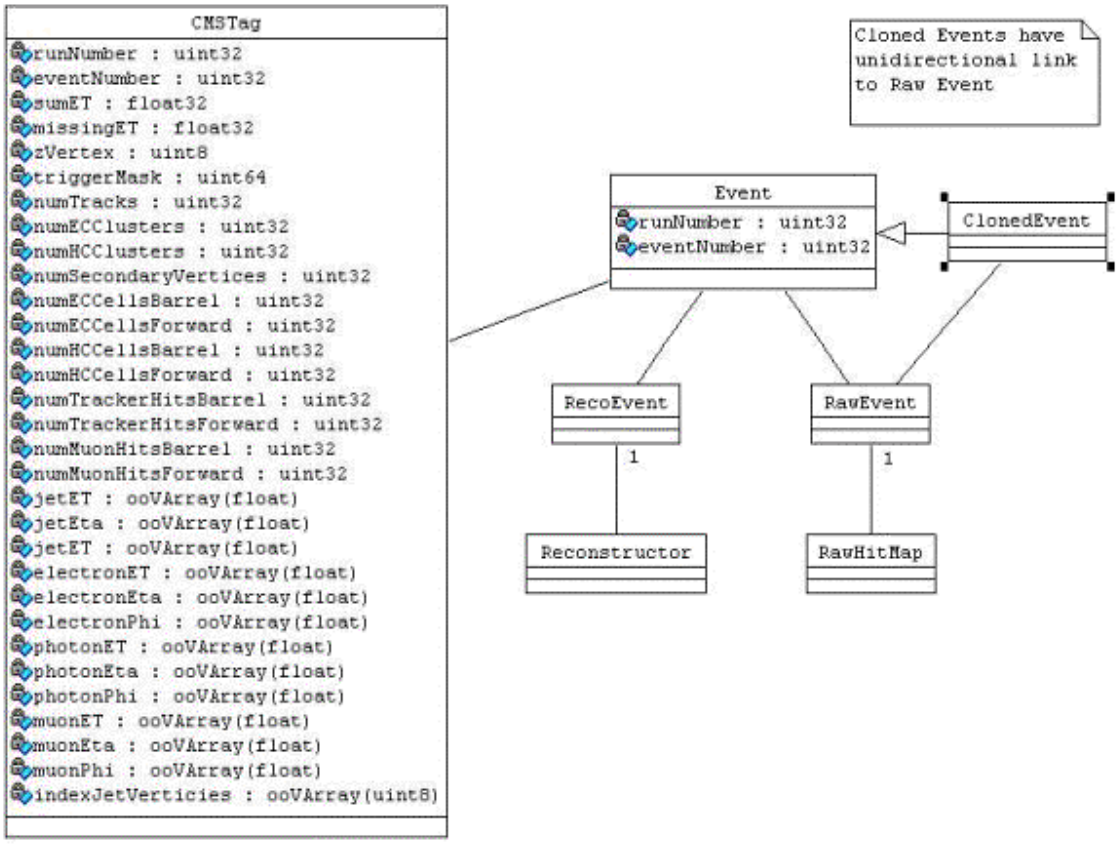


Figure 15: Showing the TAG data object and its relation to the Event classes

To process the simulated multi-jet data, the procedure used is to read the raw data files using the "ooZebra" utility developed in CMS, to create raw data objects (Tracker, Muon hit maps, ECAL, HCAL energy maps) for each event, and then to store these objects in the Objectivity database. The raw objects are used to reconstruct tracks and energy cluster objects. These new objects are in turn stored in the database. Finally, pattern matching algorithms create "physics" objects like Jets, Photons, Electrons, and Missing ET, which are subsequently stored in the database as "analysis objects".

- The detector is arranged in layers: kept as persistent objects in the DB
- Each layer contains a number of TrHits
- These TrHits are kept in a persistent class "TrHitMap" in the DB
- Reconstruction begins by selecting a trio of TrHits (★) on three layers working from the outside in.
- A transient track is generated from the trio of candidate hits and "grown" if its χ^2 is small enough
- By swimming the track, and using windows (★) on the intervening layers, unused TrHits are added to the track.
- If the track has sufficient hits, and a low enough χ^2, then it is added to a transient vector of good tracks, and the process begins again.
- Finally, an update lock is obtained on the DB, and all tracks are made persistent as ReconstructedTracks

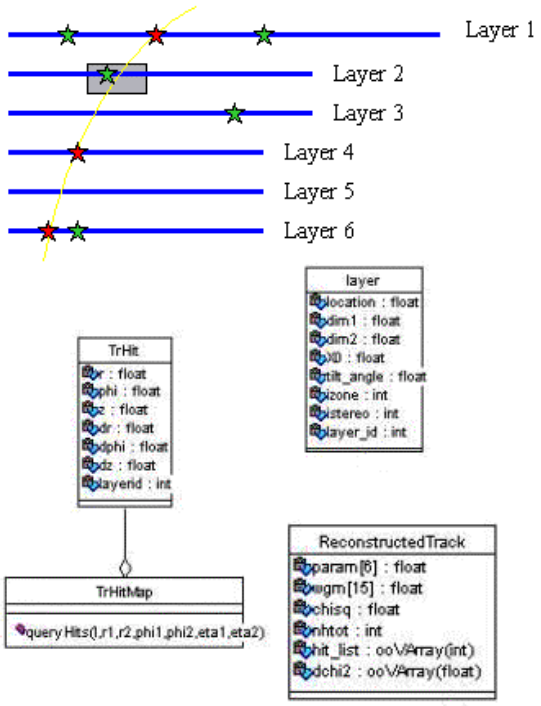


Figure 16: The procedure used in the CMSOO application to find, fit, and make persistent tracks. The method is a Kalman Filter and is based on work by Sijin Qian and Irwin Gaines.

Most of the processing to convert the raw data into objects in the database was carried out on the Exemplar, and on the HP Kayak workstation. At one point, a substantial number of events were processed on the PCSF Windows NT cluster at CERN. The configuration for this processing is shown below:

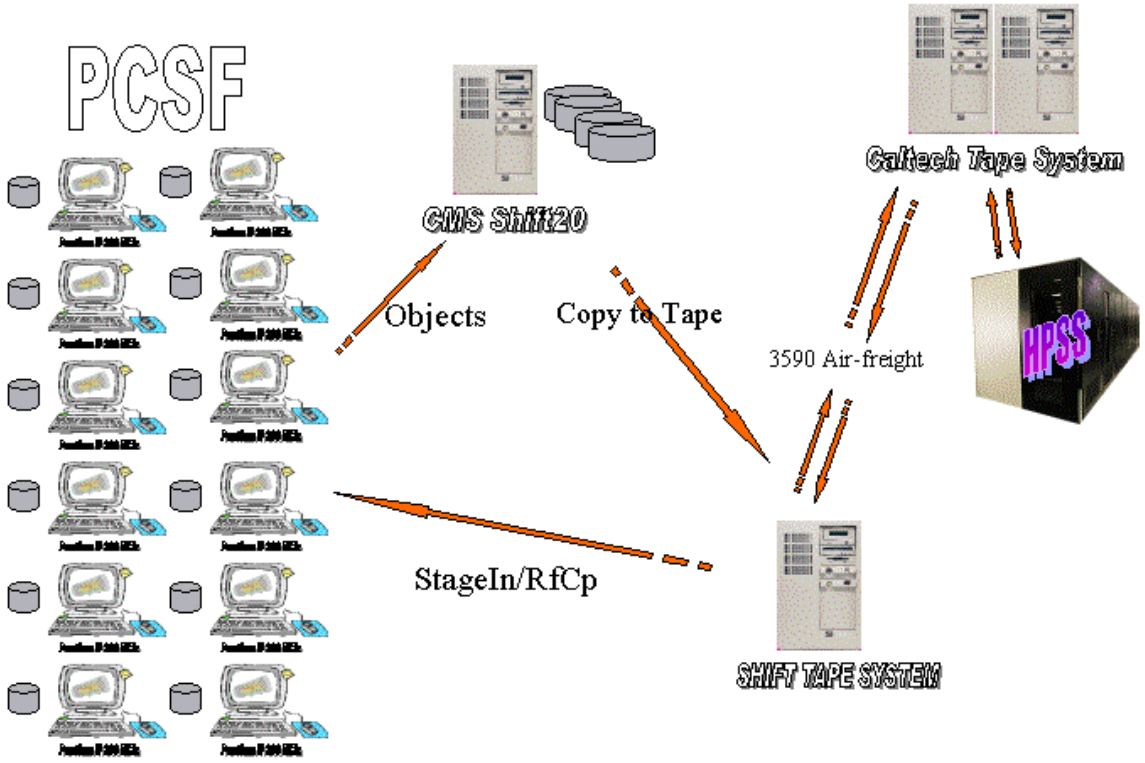


Figure 17: CMSOO processing using the PCSF cluster at CERN. The raw FZ files were shipped to CERN, and individually staged in to each of the PCSF machines. Each machine then processed the raw data into objects made persistent in an Objectivity database on a Sun.

Once the Object database federation had been populated on the Sun server (Shift20), the individual database files were ftp'ed from CERN to Caltech across the TransAtlantic link (at that time of capicity 4 Mbits/second). A total of ~35 Gbytes of files were moved in three days.

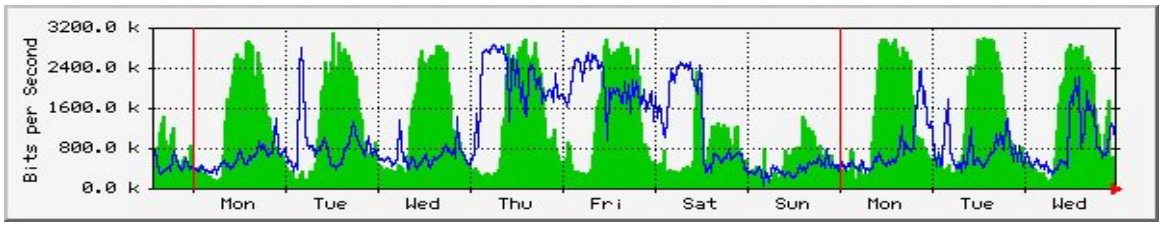


Figure 18: Showing the traffic on the HEP TransAtlantic link around the period (14/10/98) when CMSOO files were being ftp'ed from CERN to Caltech. The blue curve shows the total traffic from CERN to the USA. The CMSOO traffic can be seen as the load from Thursday until Saturday.

Accessing ODBMS data from Java

The Java API supplied with Objectivity/DB has proven to be an extremely convenient and powerful means of accessing event object data (created using the C++ CMSOO application) in the CMSOO database.

Initially, we developed 2D viewers to add us in the debugging of the CMSOO track reconstruction code.

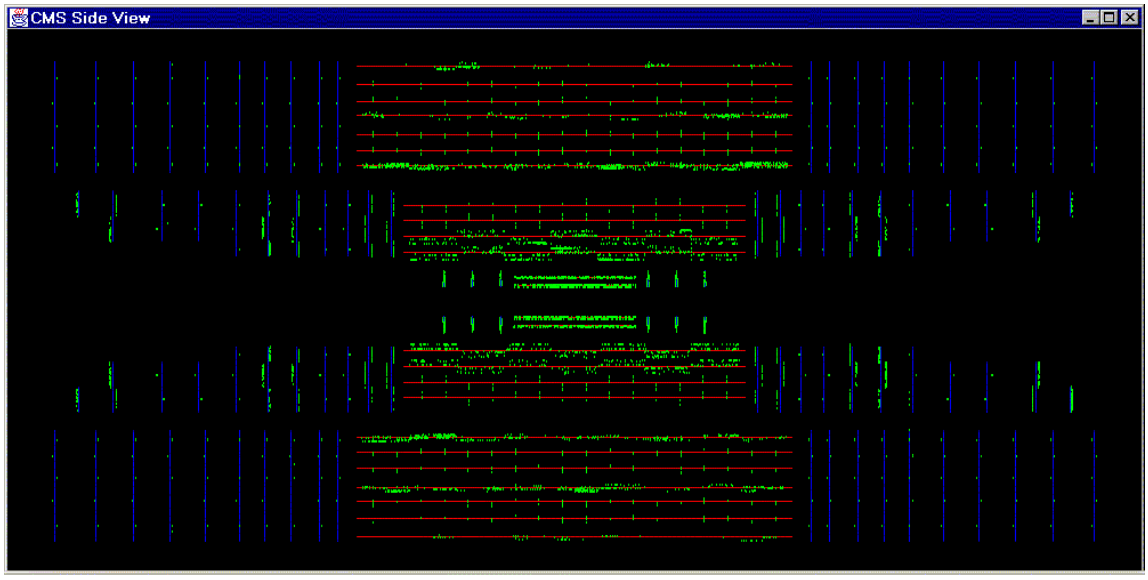


Figure 19: Showing an early prototype viewer for the CMS tracker, with accumulated “hits” from many events.

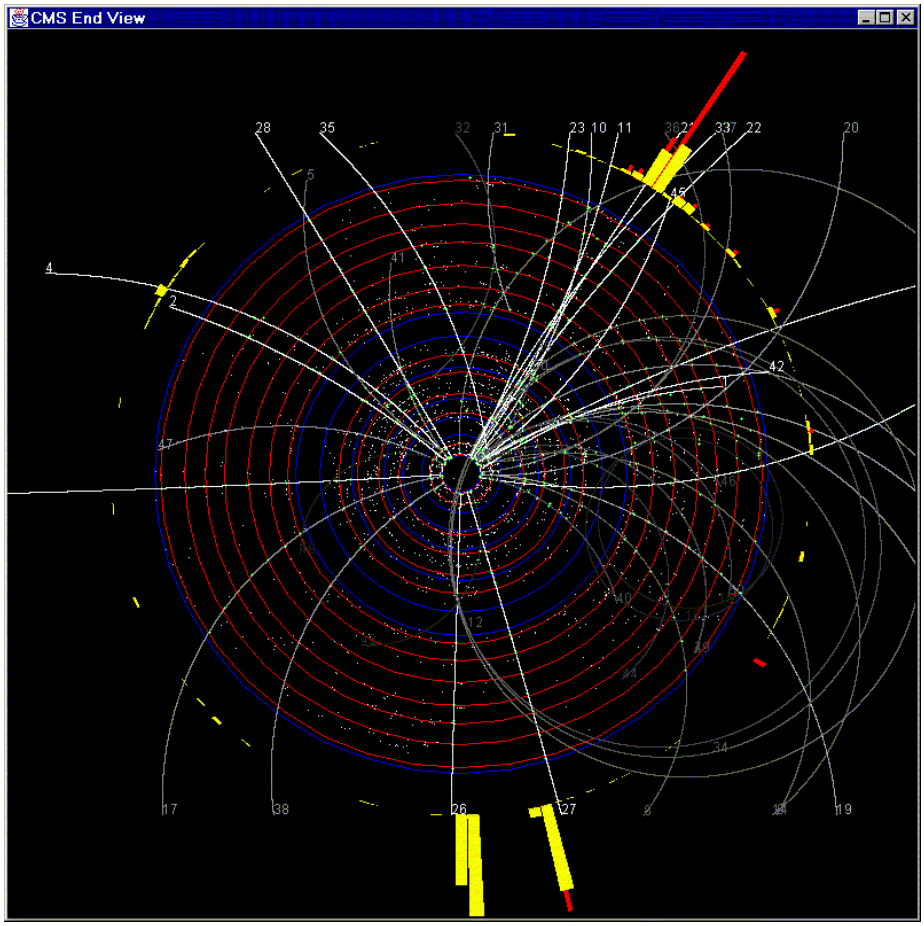


Figure 20: The prototype 2D event viewer, with tracker and ECAL information, and fitted tracks

During the course of these early developments, we uncovered several restrictions with the Java binding to the database:

- Impossible to access C++ objects from Java which contain array data members: these must be converted to Objectivity VArrays.
- Class names cannot contain an underscore character. Found to be a problem in the beta version of the Java binding.
- The "shapes" of the Java and C++ objects must match exactly, otherwise access from Java fails.
- The Applet's paint method in principle requires the geometry, hits and tracks to be re-drawn. We don't want to keep opening the database to re-access the information required to re-draw, so we need to define another method. This has been resolved by defining an off-screen image buffer, and drawing exclusively in that.

We developed a 3D event viewer, which directly fetches the CMS detector geometry, raw data, reconstructed data, and analysis data, all as objects from the database. A screen capture from the event viewer is shown in the Figure below.

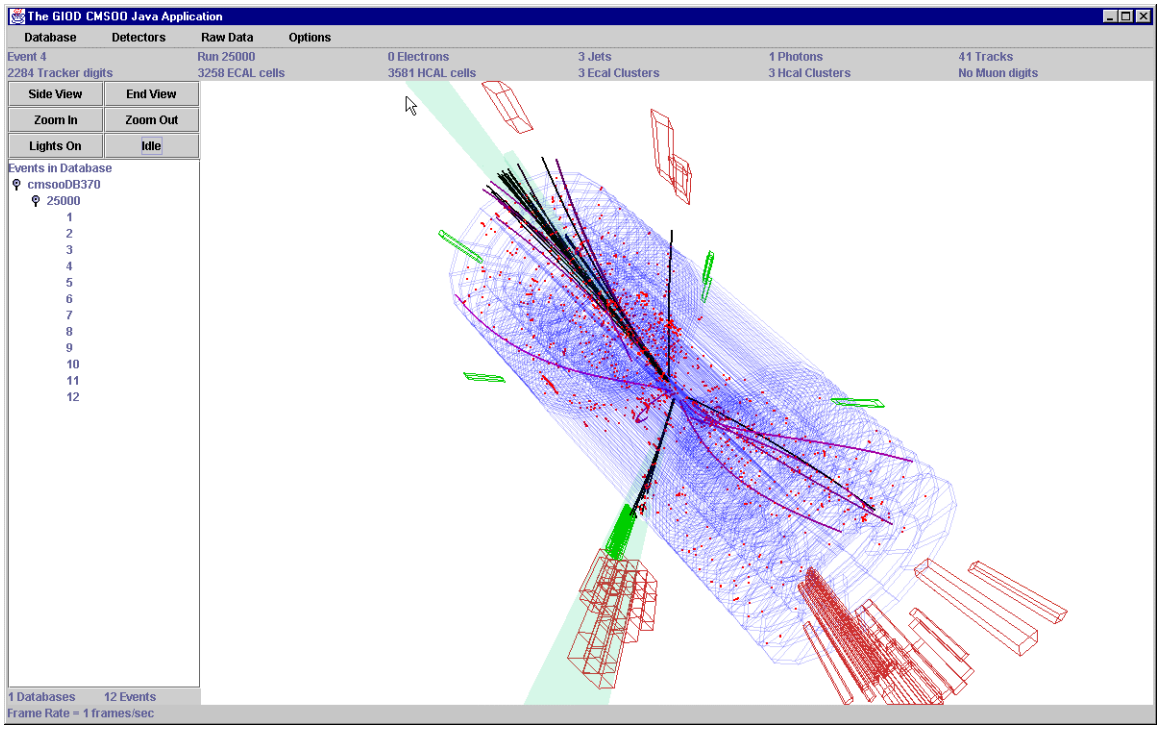


Figure 21: The JFC/Java3D-based GIOD Event viewer

In addition, we have used SLAC's Java Analysis Studio (JAS) software, which offers a set of histogramming and fitting widgets, as well as various foreign data interface modules (DIMs). Using JAS, we constructed a DIM for Objectivity, and a simple di-Jet analysis routine shown below:

Table 5: The simple di-jet analysis procedure used in JAS for the CMSOO data

```
public void processEvent(final EventData d) {
   final CMSEventData data = (CMSEventData) d;
   final double ET_THRESHOLD = 15.0;
   Jet jets[] = new Jet[2];
   Iterator jetItr = (Iterator) data.getObject("Jet");
   if(jetItr == null) return;
   int nJets = 0;
   double sumET = 0.;
   FourVectorRecObj sum4v = new FourVectorRecObj(0.,0.,0.,0.);
   while(jetItr.hasMoreElements()) {
       Jet jet = (Jet) jetItr.nextElement();
       sum4v.add(jet);
       double jetET = jet.ET();
       sumET += jetET;
       if(jetET > ET_THRESHOLD) {
           if(nJets <= 1) {
              jets[nJets] = jet;
              nJets++;
           }
       }
   }
   njetHist.fill( nJets );
   if(nJets >= 2) {    // dijet event!
       FourVectorRecObj dijet4v = jets[0];
       dijet4v.add( jets[1] );
       massHist.fill( dijet4v.get_mass() );
       sumetHist.fill( sumET );
       missetHist.fill( sum4v.pt() );
       et1vset2Hist.fill( jets[0].ET(), jets[1].ET() );
   }
}
```

With the analysis routine, we were able to iterate over all events in the CMSOO database, apply cuts, and plot the di-jet mass spectrum for the surviving events. The following Figure shows the JAS di-jet mass histogram.

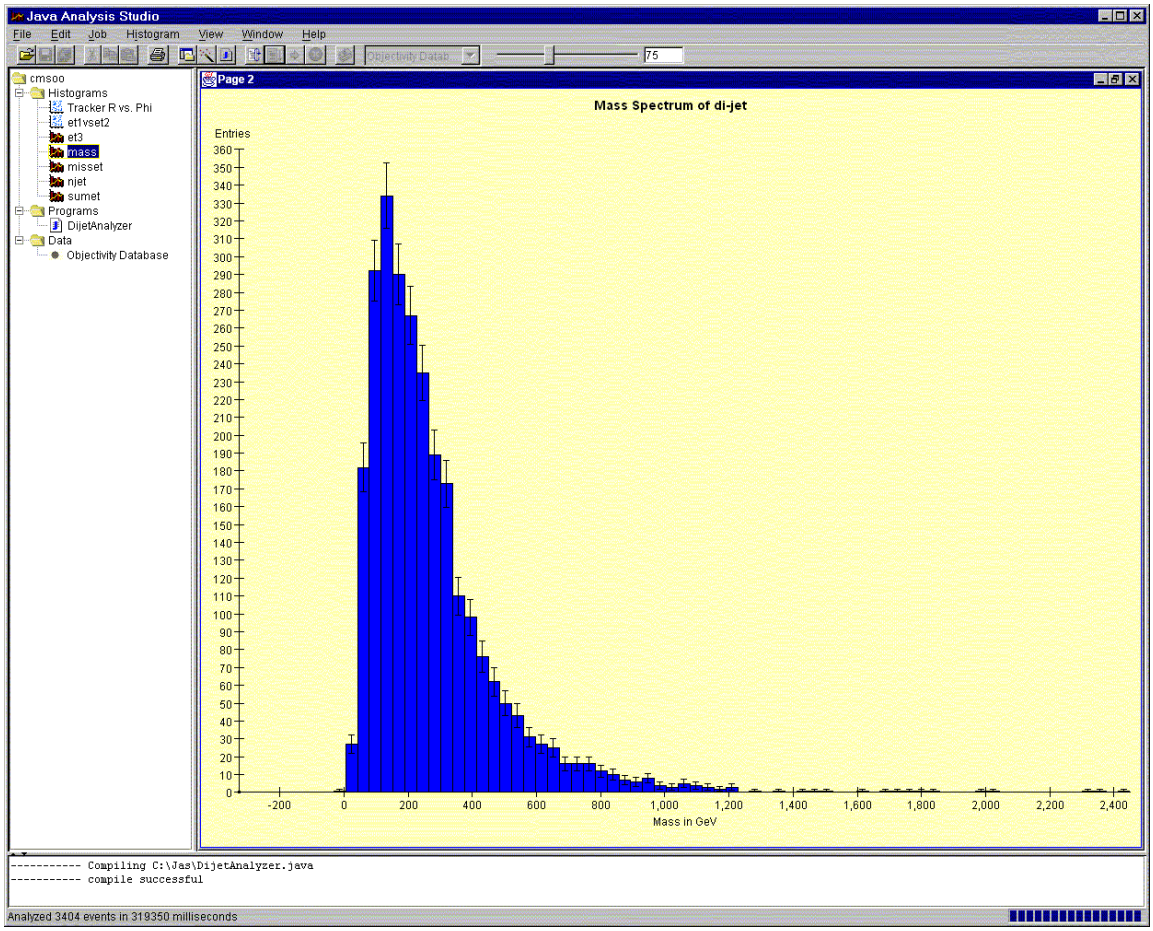


Figure 22: : Showing SLAC's Java Analysis Studio with the GIOD Objectivity DIM, used to analyse a set of di-jet events from the CMSOO database, and plot the di-jet mass spectrum.

Pure Java Track Fitter

We have developed a demonstration track fitting code in Java, that efficiently finds and fits tracks with Pt > 1 GeV in the CMS tracker. The code identifies good tracks at a rate of ~ 1 per second, for a total set of ~3000 digitisings in the tracker. This compares favourably with the C++/Fortran Kalman Filter code we use in our production reconstruction code (which also operates at about 1 track per second, but which is a considerably more compute intensive procedure).

The image below shows a single di-jet event in the CMS calorimeter, with the reconstructed tracks from the Kalman Filter in white/grey, and the reconstructed tracks from the Java fitter in orange. We have tested both codes on simulated single muon event samples at energies of 1, 2, 5, 20 and 50 GeV. Single track finding efficiencies for both codes on these events is excellent (a high statistics study is planned).

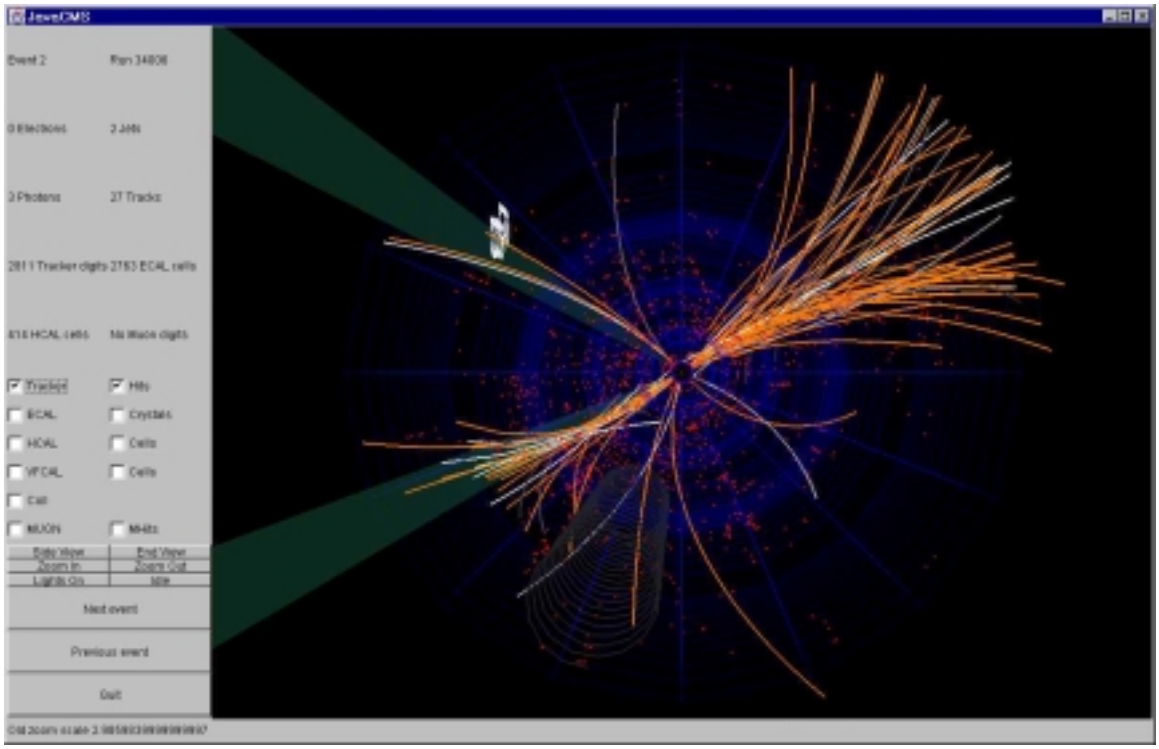


Figure 23: The Java track fitter: in the event shown there are 27 tracks found by the Kalman Filter, and 58 found by the Java fitter

Fitting technique

- The tracker digits are fetched from the Objectivity database as a set of space points (or hits) in (r,phi,z) with associated errors.
- The hits are arranged in layers corresponding to the Tracker detector layers.
- Starting from the outermost layers of the detector, a pair of seed hits are chosen that line up roughly with the centre of CMS.
- Working from the innermost layer of the detector, a third seed hit is chosen that, when coupled with the two initial hits, can be fitted to a helix with a radius corresponding to a momentum of at least 1 GeV, the fit giving an acceptably small Chi squared.
- Using the candidate track fit, and working from the innermost layer of the detector, the track is extrapolated to each layer of the detector it intersects, in order to build a list of candidate hits on the track.
- At each intersected layer, the closest hit is determined, and the hit added to the track list if its Chi squared distance to the track intersection point is below a certain value
- If an acceptable hit is found on the intersected layer, then the Chi squared is accumulated for the track
- If the accumulated Chi squared of the track at any time exceeds a certain value, then it is invalidated, and a new trio of seed hits sought
- After the candidate track has been extrapolated through the detector, it is rejected if it does not meet the following conditions:
    - it has at least three well-measured hits in its track list (i.e. hits from Stereo or Pixel layers)

    - it has at least six hits in total in its track list
    - it's Chi squared is below a certain value
- If the track meets these criteria, then the innermost three well-measured hits are used in a new three point fit to a helix. This fit yields the parameters of the helix, and a Chi squared. The track is only accepted if this Chi squared, in turn, is less than a certain value.
- For an accepted track, all hits in the track list are marked as "used", and fitting resumes for further tracks with a new three point seed.
- Once all tracker layers have been iterated over, or once all hits have been marked "used", fitting terminates.

Note that, unlike the Kalman Filter method, this fitting procedure does not give the covariance matrix for the fitted track parameters, nor does it take into account multiple scattering in the detector, not does it use all points on the track in the final fit. These deficiencies will be addressed in further work.

Further work planned on the Java Fitter and Reconstruction

Using this fitter as a basis, we had plans to develop a full Kalman Filter fitter in Java. This would have been fully integrated in our JavaCMS event viewing applet, allowing the user to interactively refit existing tracks, or fit new tracks. The new fitter would also have been integated in a completely Java-based reconstruction tool for our CMSOO database.

ATM traffic from database clients

Recent work in GIOD has been focussing on the network-related aspects of using the CMSOO database. Tests have begun which involve distributing a number of database client processes on the Exemplar, and having them communicate with a database hosted remotely across a dedicated ATM fiber link on an HP C200 workstation. We have measured the I/O throughput to the disk containing the database files, the I/O traffic on the ATM network, and the load on the database host processor during the tests. At the time of writing, the test results are still being interpreted, but an initial clear conclusion is that the Objectivity database lock and page servers play important roles in governing the maximum throughput of the system.

Multiple client tests

The system setup is shown below. The HP/UX system is a **C200**, running Objectivity v5.0. The data disk is a narrow SCSI device, rated at ~8 MBytes/sec. The ATM is on dedicated fibre (no other users), running at 155 Mbits/sec.

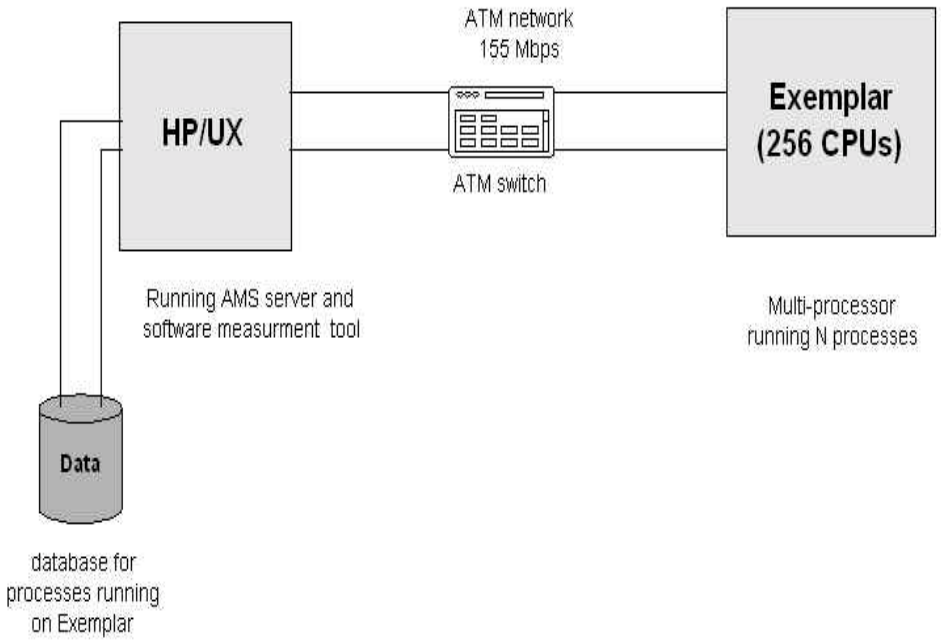


Figure 24: The layout of the system for the ATM tests

Test 1 : Transfer of ~40 large files using FTP

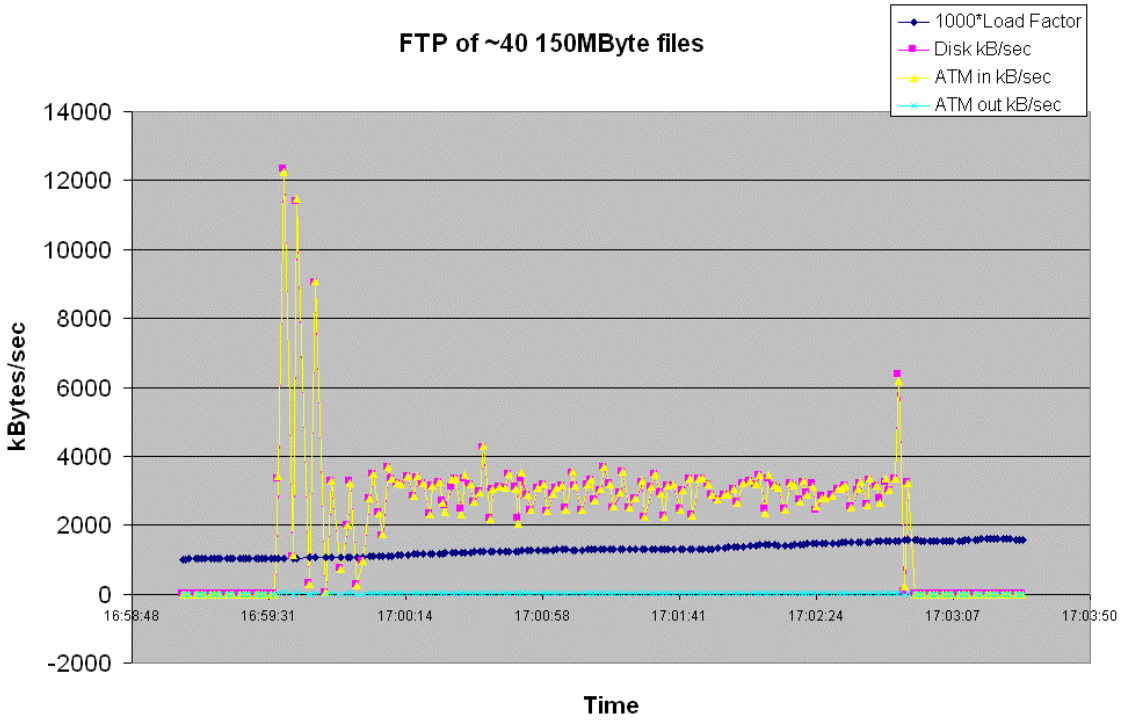


Figure 25: FTP of a large number of large files from the Exemplar to the C200 Disk.

Remarks:

- The "Load Factor" is the value obtained using the "uptime" command on the C200
- The disk rate is obtained using the "du -ks" command on the C200 and dividing by the elapsed time
- Note that the initial rate obtained of ~12 MBytes/sec on both the ATM and the disk was confirmed by the ftp utility, which reported 11.5 MBytes/sec for the second file transferred.
- Why this exceeds the nominal maximum rate for the SCSI disk is not understood.
- The oscillation in rates at the start of the transfers is not understood.
- The ATM rates are obtained by using the "atmmgr 0 show -c" command on the Exemplar to obtain the cell count in and out of the adapter, using a cell size of 47 bytes, and dividing by the elapsed time.
- This value of the cell size is confirmed for this payload by measuring the cell count for the ftp of a file of known size in bytes.

Test 2: CMS Event Reconstruction in one processor.

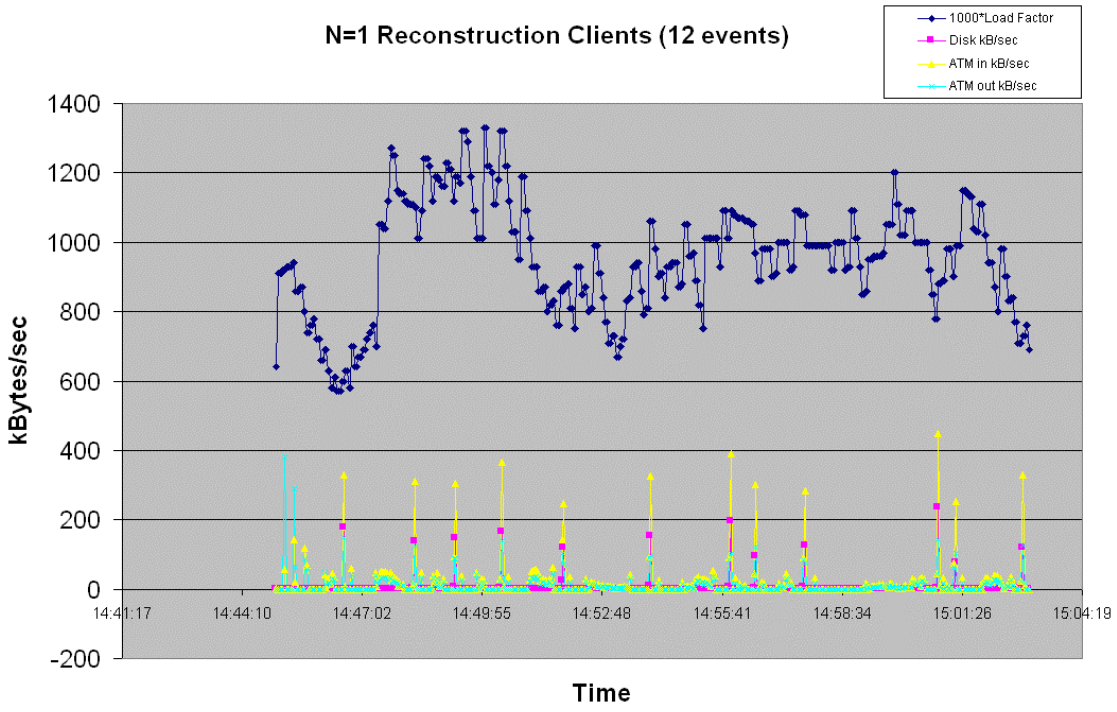


Figure 26: Network traffic for a single Reconstruction client

Remarks:

The traffic destined for the database is exclusively object data, and consists of "raw" event data (like hit maps), "reconstructed" objects (like tracks) and analysis objects (like jets), and also small "tag" objects.

Note that the Reconstruction program processes 12 separate events: hence the 12 yellow spikes in input ATM traffic in the graph.
Note that the write rate to the disk is always lower than the input rate from the ATM.
There is significant traffic **towards** the Reconstruction client, particularly when the client starts up: this is the object schema information sent by the server to the client.

There is a lot of traffic in both directions on the ATM, but a small fraction of it ends up as objects in the database.

It is not understood what the excess traffic consists of.

The Load Factor on the C200 seems to fluctuate oddly (compare with the very smooth behaviour during the ftp transfer).

The cause of this is not understood. Perhaps either the AMS or LockServer ... requires further tests.

Test 3: CMS Event Reconstruction in 16 processors.

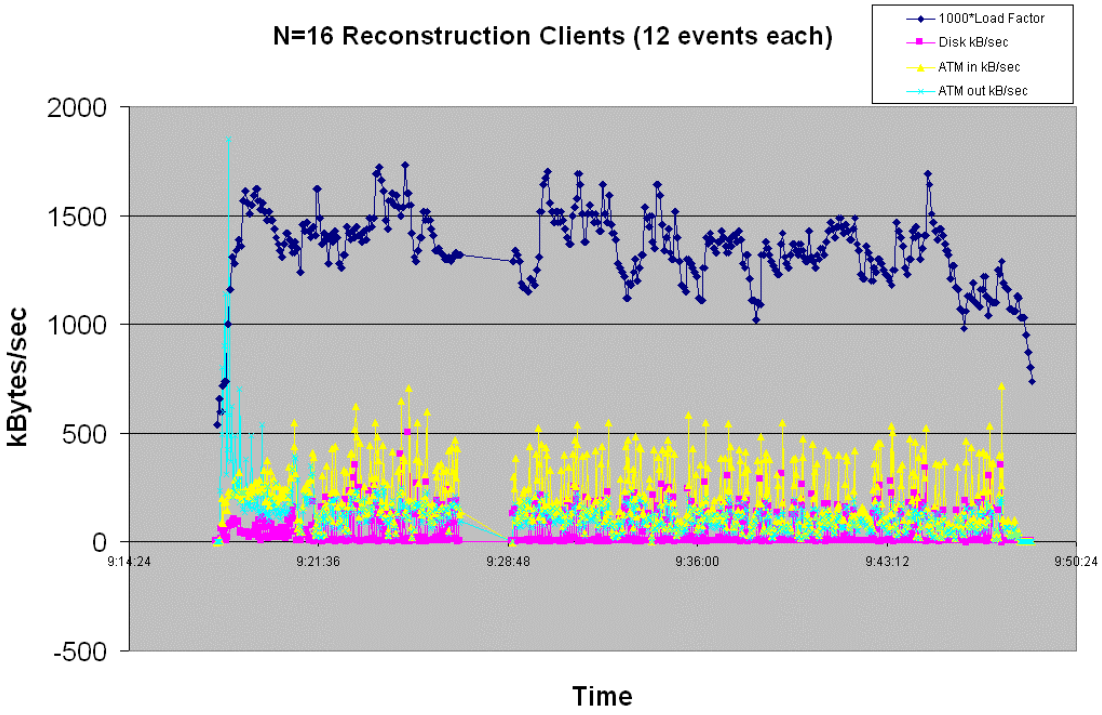


Figure 27: Network traffic caused by 16 simultaneous Reconstruction clients

Remarks:

Again, very high traffic rates **towards** the clients when they start up.
Note that the Load Factor is now ~1.5 on the machine (cf ~1.0 for a single client).

ATM and TCP/IP Characteristics of the Database Client traffic

The unit of transfer between the TCP software on two machines is called a segment. Usually, each segment travels across the network in a single IP datagram. Each segment is divided into two parts, a TCP header followed by data. The organization of the header is shown below.

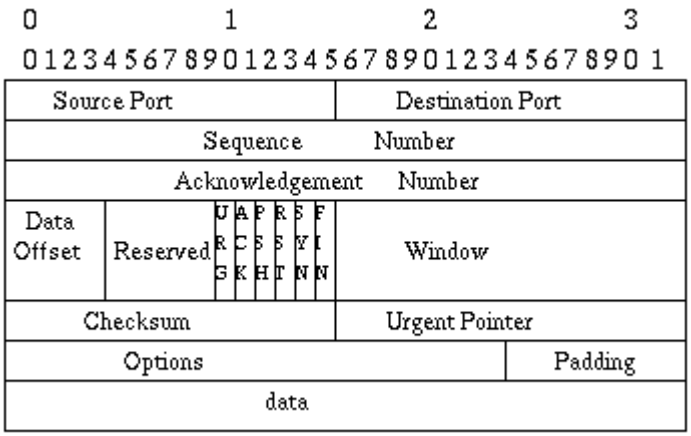


Figure 28: TCP Header format

The source port and destination port fields identify the end points of the connection. The sequence number identifies the first data octet in this segment. The exception is that when SYN is present, the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1. If the ACK control bit is set, the acknowledgement field contains the value of the next sequence number the sender of the segment is expecting to receive. Note that the sequence number refers to the stream flowing in the same direction as the segment, while the acknowledgement number refers to the stream flowing in the opposite direction as the segment. The TCP Data Offset (also called Header Length) tells how many 32-bit words are contained in the TCP header. This information is needed because the Options field has variable length, so the header length is variable too. Reserved has 6 bits and must be zero. They are for future use. Some segments carry only an acknowledgement while some carry data. Others carry requests to establish or close a connection. TCP uses the six 1-bit flags to determine the purpose and contents of the segment. URG is set to 1 if the Urgent pointer is used to indicate a byte offset from the current sequence number at which urgent data are to be found. It's used in interrupt messages. ACK bit is set to 1 if the acknowledgement field is valid. PSH bit causes the remote TCP layer to pass this segment immediately to the application layer without waiting to form a larger buffer. The SYN bit is used to establish connections. The FIN bit is used to release a connection. It specifies that the sender has no more data. After closing a connection, a process may continue to receive data indefinitely. The RST bit is used to reset a connection. It indicates that an error has occurred and the connection should be forcibly closed. A 16-bit field Window (also called sliding window) in the TCP header tells how many bytes may be sent beyond the byte acknowledged. It's used in flow control. Like Acknowledgement, a checksum is provided for reliability. The checksum algorithm is simply to add up all the data, regarded as 16-bit words, and then to take the 1's complement of the sum.

Throughput

By drawing throughput distribution graphs, we can observe and analyze network traffic using statistical methods. If packets (often from more than one segment) or in this case an ATM cell is b bytes long and the time it takes the packets to traverse from one end to the other is T seconds, then

$$\text{throughput} = \frac{b}{\triangle T}$$

For the dedicated ATM line from the C200 to the Exemplar, the value b is about 47.625±0.002 bytes per cell (obtained with repeated FTP transfers of about 500 Megabytes of data). This measure is the direct result from the fact that one ACK can acknowledge more than one segment.

All measurements on the C200 are conducted with a c-shell script which polls the network periodically with the tool 'atmmgr'. The command 'atmmgr' gives a direct packet count on the ATM line at any given moment. Disk measurements are currently conducted with the 'du' command. Since 'du' takes about 3 milliseconds on the C200, the measured disk rate and the ATM rates are offset from each other by that amount. This is insignificant, however, compared to the polling period (1 to 2 seconds).

Network Monitoring Tool:

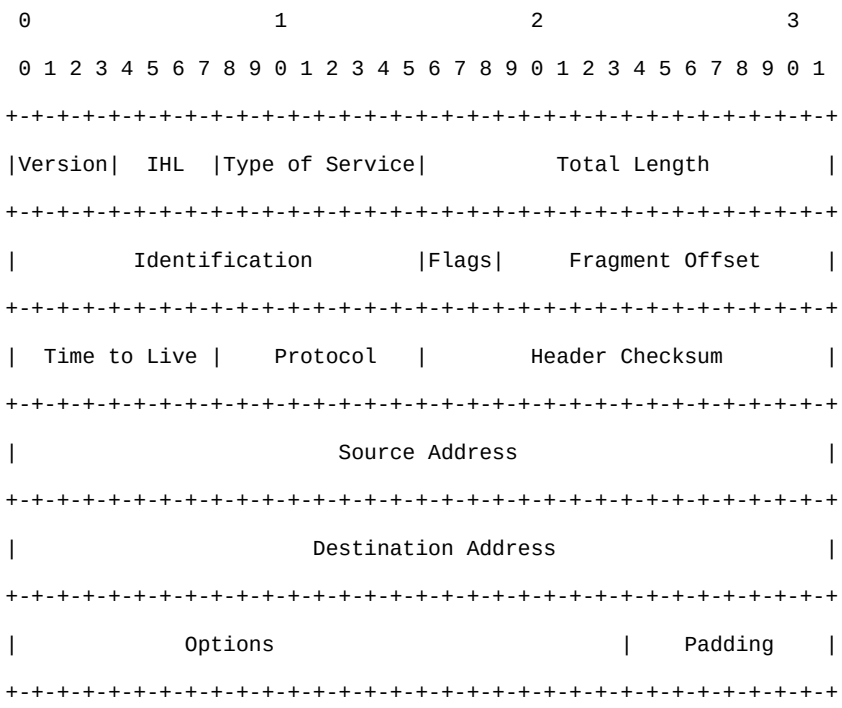
We implemented a new tool, termed 'Monitor', to compute and display more meaningful information from the raw data collected by a program called 'tcpdump'.

Packet capturing can be used for much more detailed analysis of network data. There is a really nice public domain packet capture tool called 'tcpdump.' This tool will generate a gzipped tar ball of raw data, which can then be reconstructed (with an appropriate script or a program). For instance, we can calculate the precise elapsed time, bytes/segments sent and received, retransmissions, round trip times, window advertisements, instantaneous throughput, precise measurement of the RTT, etc.

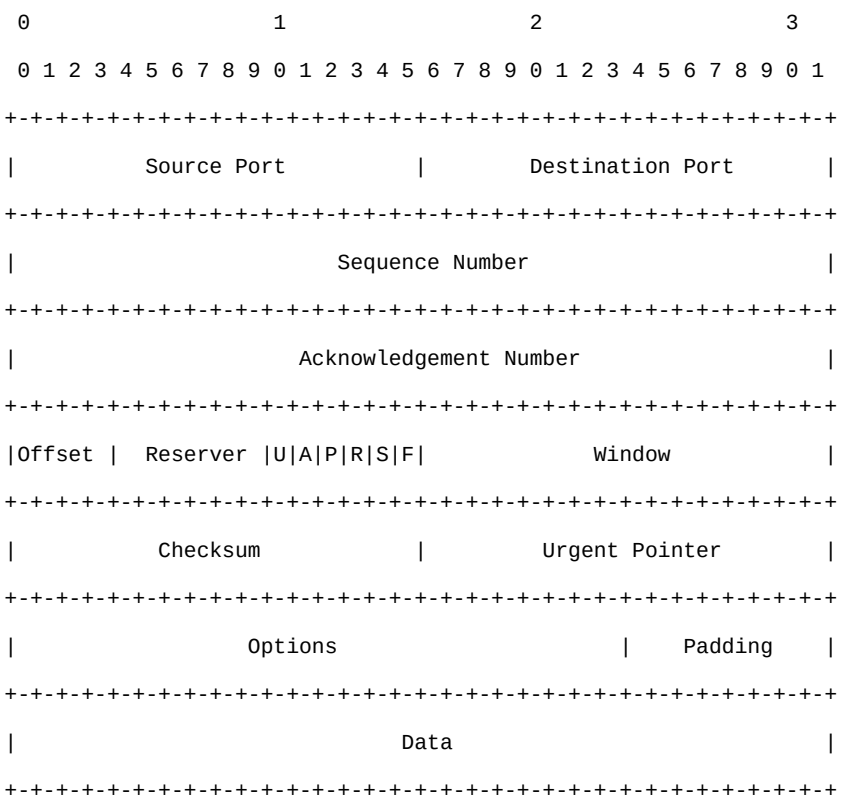
There are distinct advantages of having both the instantaneous throughput as well as the rough estimates made by simply counting the packet numbers received. By drawing throughput distribution graphs for both, we can observe and analyze network traffic using statistical methods. If packets (often from more than one segment) are b bytes long and the round-trip time (RTT) is T seconds, then this measure is the direct result from the fact that one ACK can acknowledge more than one segment, or throughput = b/ T. The other, more precise measure is the "instantaneous throughput". It is defined as the size of the segment that caused it divided by the time elapsed since the previous segment arrived or was sent.

Packet Formats

IP (RFC 791)



TCP (RFC 793)



The “Monitor” java application for analyzing tcpdump output

'Monitor' generates graphs from the raw data collected by tcpdump.

- Time Sequence Graph - shows packets of segments sent and ACKs returned as a function of time (packet size vs. time).
- Instantaneous Throughput - shows the instantaneous (averaged over a few segments) throughput of the connection as a function of time.
- Autocovariace - shows autocovariance coefficient versus lag graph.
- Round Trip Times (not yet implemented) - shows the round trip times for the ACKs as a function of time.

'Monitor' is implemented in Java JDK 1.1. More specifically, the interface is written with the AWT widget set. 'Monitor' is implemented in one main class and four sub classes, listed in hierarchal order.

Monitor.java - The main class which reads a data file, calculates and plot the statistics(described above) in time series, histogram, and zoomed. It accept the standard tcpdump data file.

FigureFrame.java - The class implementing the graphics frame. It draw the axis, scales, titles, labels and coordinates around its figure, with a GraphArea object held in the center. It can also generate subfigures of histogram, autocovariance and zoom in.

GraphArea.java - The class which implements the graphic area where the plots take place.

MyDialogBox.java - This class is used to implement the dialog box for selecting the data files. The Java Tutorial says that: "Because no API currently exists to let applets find the window they're

running in, applets generally can't use dialogs." MyDialogBox use Frame as Dialog, and keeps a handle of its parent applet in "parentApplet" for input parameters and actions

AboutBox.java - The class implementing the about/help dialog box. It simply provides basic documentation for the end user.

Shown below are screen shots from Monitor.

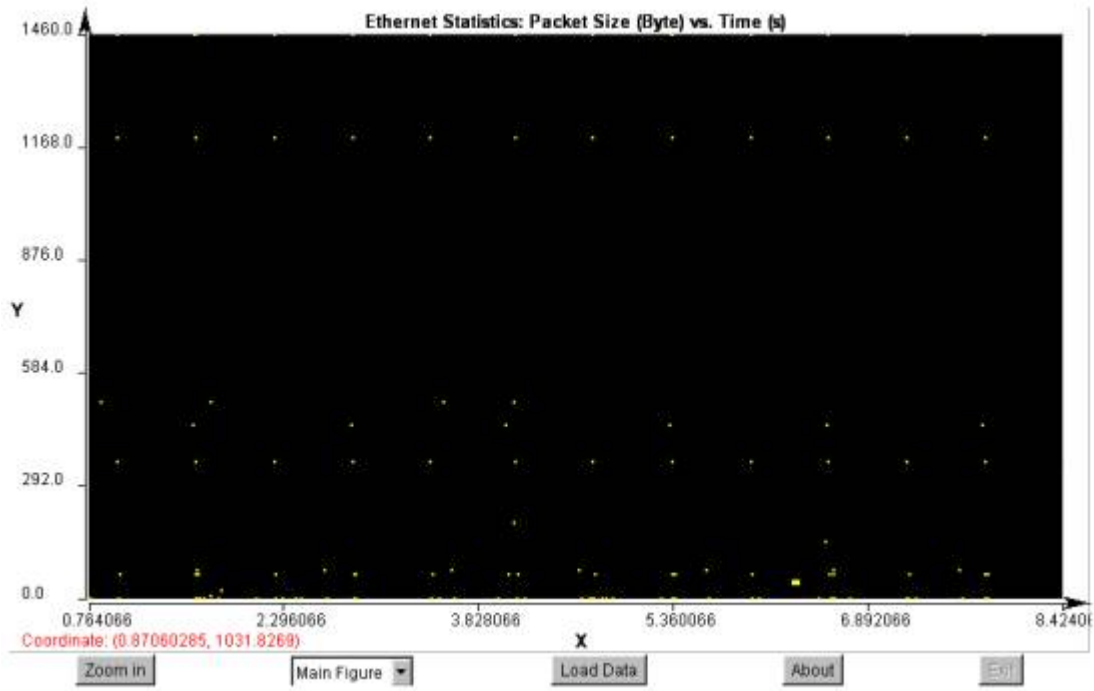


Figure 29: Screen shot from the "Monitor" program

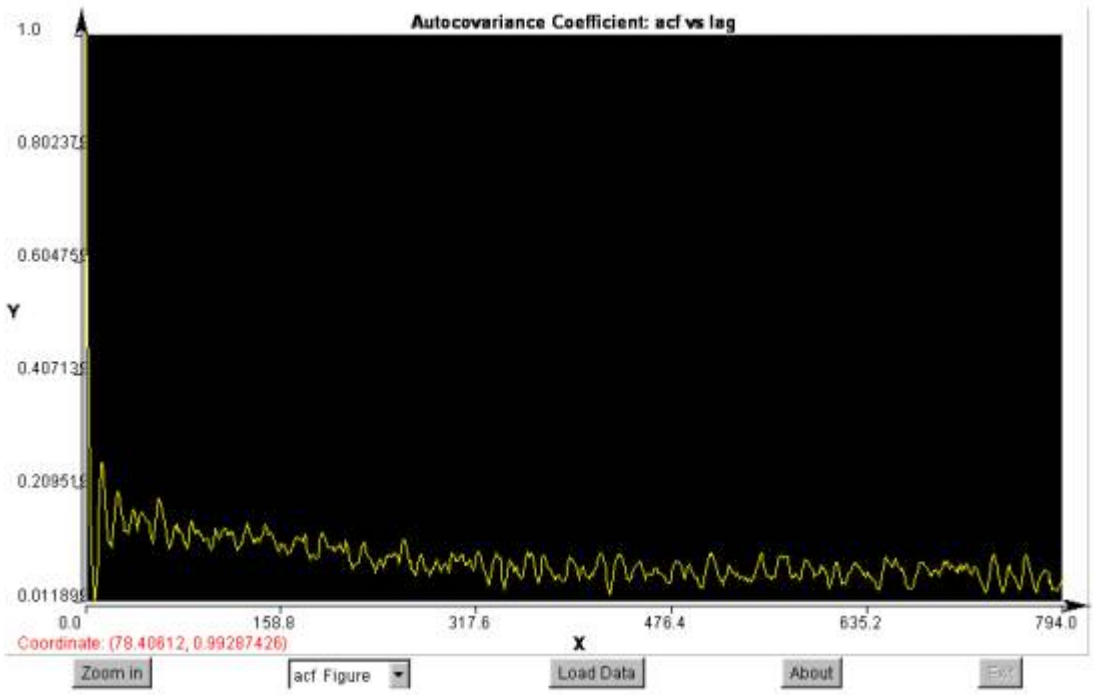


Figure 30: Screen shot from the "Monitor" program

FTP Tests through the HP-C200.

The first graph shows initial FTP tests with the smaller (20 Kbytes) packet size. Note that the disk rates can become negative due to the resizing of the cluster size. Initially a larger cluster (in 256k inodes) is reserved for the data, and then later resized to reflect the actual amount of data being s

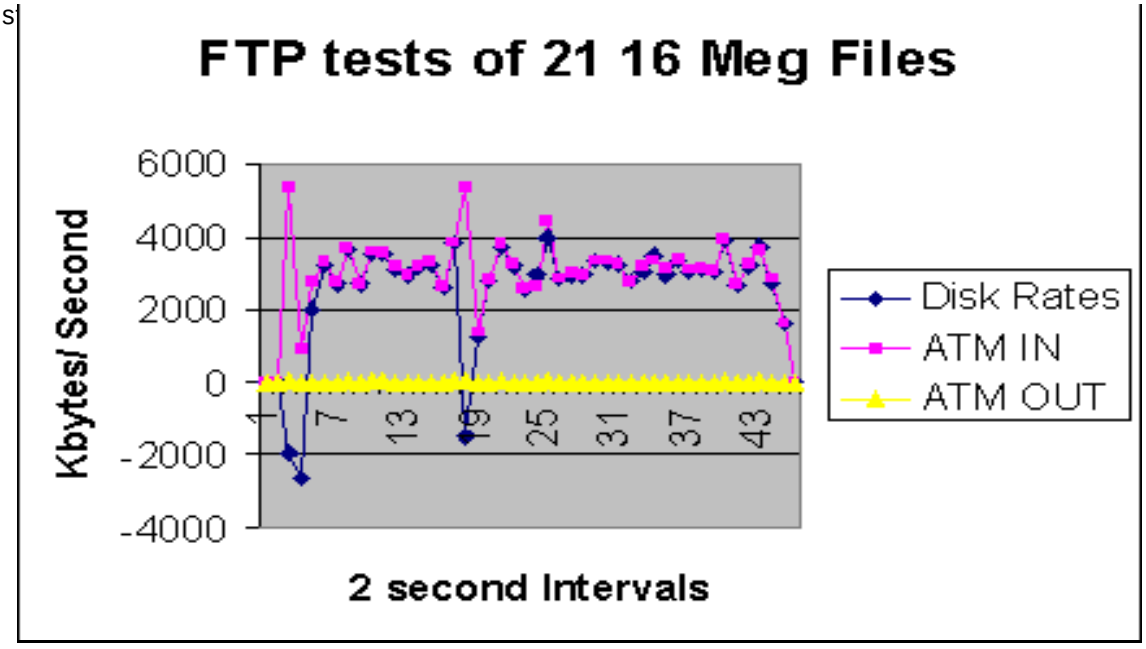


Figure 31: Traffic profile for the FTP of 21 files with the small packet size

FTP performance through this ATM is fairly high, with an average transfer rate of about . The spiking behavior is due to both the TCP protocol and the behavior of the operating system. Using small packets can adversely affect single workstation-to-server file transfer speed. To enhance performance, the use of larger packet sizes can increase throughput dramatically. However, the drawback is that stagnation and network overloading can occur more easily when multiple clients are interacting with the server. The following graph shows the FTP tests repeated with the larger packet size (200 Kbytes).

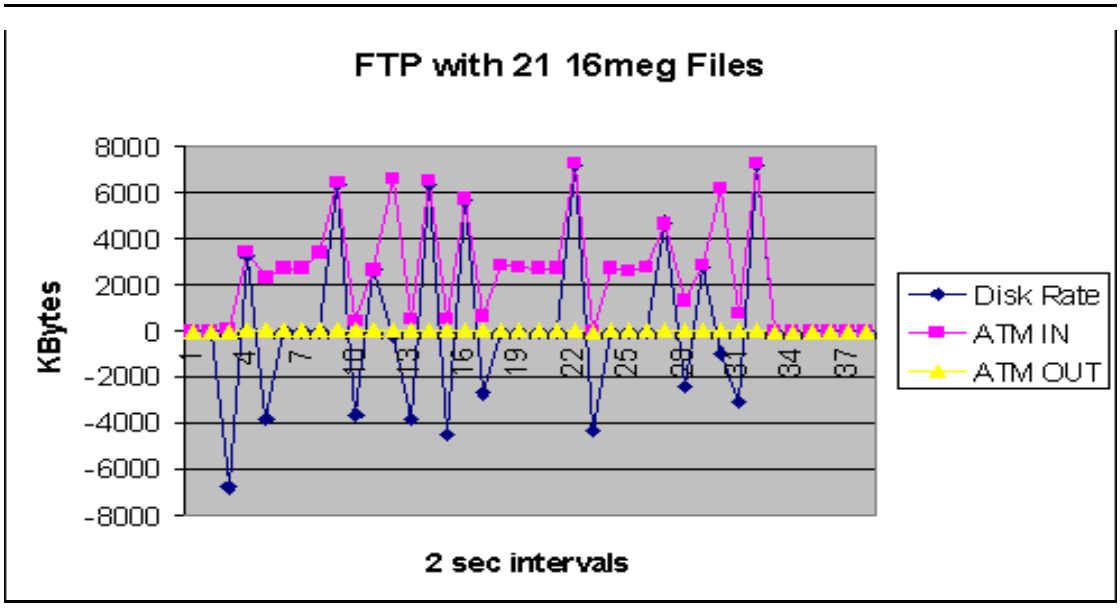
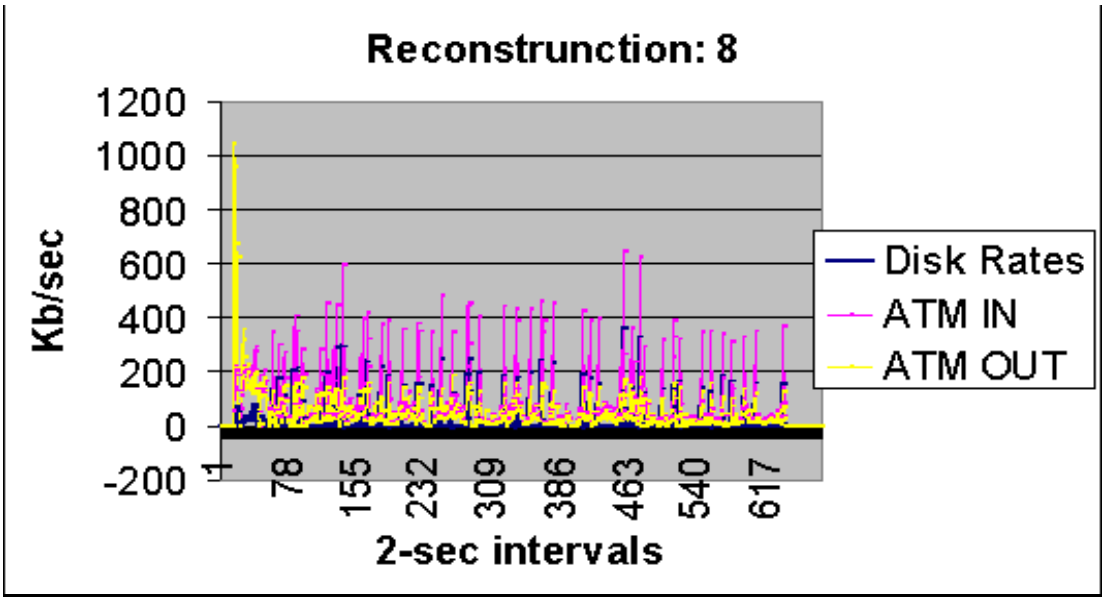


Figure 32: Traffic profile for the FTP transfer of 21 files with a large packet size

As shown above, the larger packet size improved downloading speed by about 30%. This optimization, however, has less of an effect in terms of running GIOD processes, since Objectivity divides its processes into segments which are smaller than 20 Kbytes.

Reconstruction with N clients

Reconstruction tests were performed for 1, 2, 4, 8, 16, and 32 jobs. Typical graphs are shown below.



F

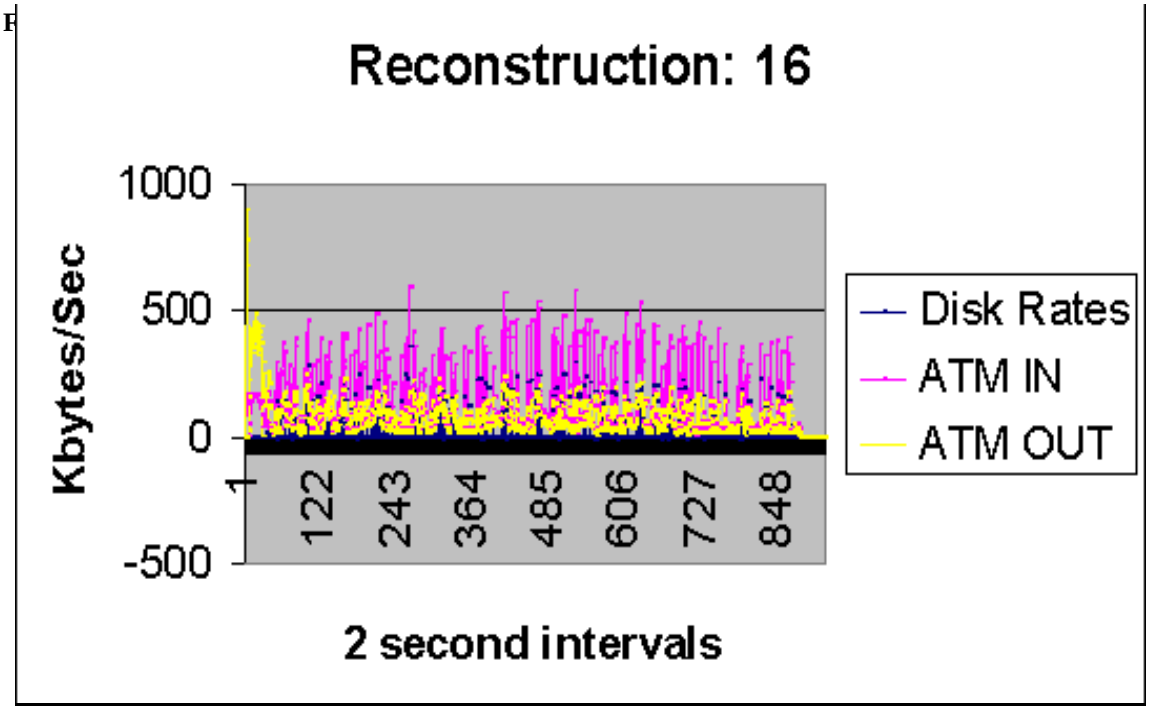


Figure 34: With 16 simultaneous Reconstruction clients

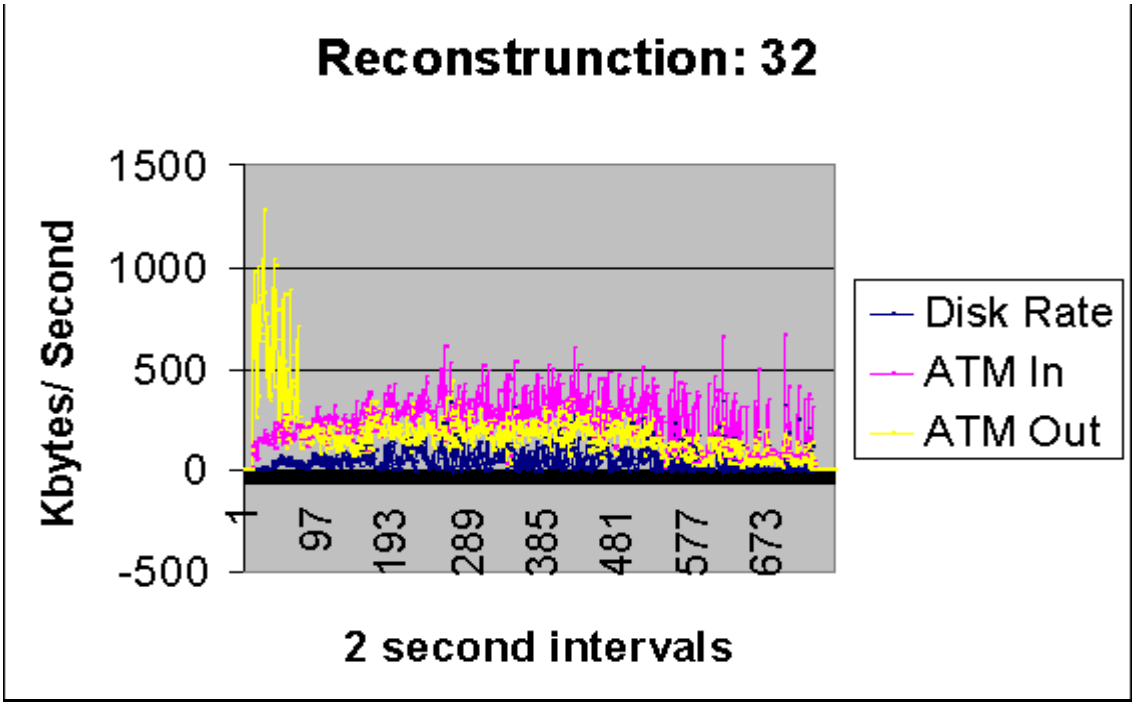


Figure 35: With 32 simultaneous Reconstruction clients

We note that the number of spikes increase roughly linearly with a larger number of clients. The receiver gives a variable size sliding window in the acknowledgment packet. Sliding window specifies how much new data the receiver could accept from the sender and makes it possible for the sender to send multiple segments before an acknowledgment arrives. If multiple segments are received at a node before an acknowledgement is ready to be sent, it is possible to acknowledge just the last segment received, so reducing the need to acknowledge every segment. This allows higher throughput and more efficient use of the bandwidth. Sliding window reflects the buffer available at the receiver. If the window values are often low or reach zero in the middle of data transfer, a device does not have enough memory allocated to buffers. A receiver may validly reduce the window size to zero to stop the flow of data completely, that is, to invoke flow control. The flow control causes the spiking behavior exhibited in the graphs above. With a greater number of simultaneous jobs, the ATM line has insufficient buffering, which in turn causes more spiking. The condensed spikes actually increase efficiency in this case.

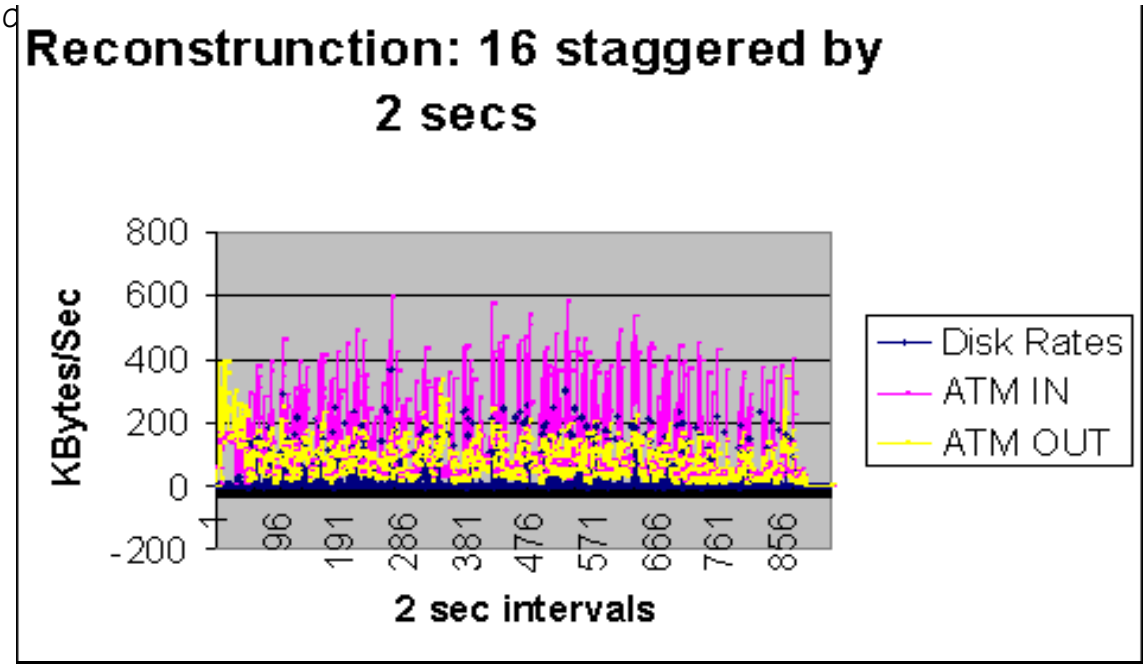


Figure 36: New Graph for Reconstruction 16 after staggering the start of processing.

Note that at the beginning of the reconstructions (original graph), the outgoing rate through the ATM line increases exponentially. This is caused by the simultaneous fetching of a list of reconstruction jobs by the clients. While this is perfectly acceptable (i.e., below saturation point) on our dedicated ATM line, it could pose major problems on slower networks. In the new graph above, the initial start of the individual reconstruction processes are staggered by delaying the start of the N^{th} process by 2xN seconds. We see that, the result is that the ATM line is less saturated. In practice, of course, clients are unlikely to be fetching jobs all at once. In such a case, normal congestion control uses the sliding window to provide for flow control within the network between them, but the problem comes when the network between the nodes is congested. If the sender has no knowledge of this, it will send the maximum amount of data allowed by the receiving node and in doing so, cause even more congestion. To overcome this, a mechanism called "slow start" is used. This keeps account of the unacknowledged segments in what is referred to as the congestion window. At the start of a connection the value of the congestion window is one segment. If the first segment is acknowledged, the window is then increased in size; if retransmission occurs, the congestion window is then decreased in size. So the congestion window keeps track of the state of the links between two nodes so as not to overload them.

FTP Tests with the San DiegoSuper Computer Centre (SDSC)

The San Diego SP2 is connected to the Exemplar via a combination of ATM, FDDI, and wide area Ethernet. The configuration for our tests is show schematically below:

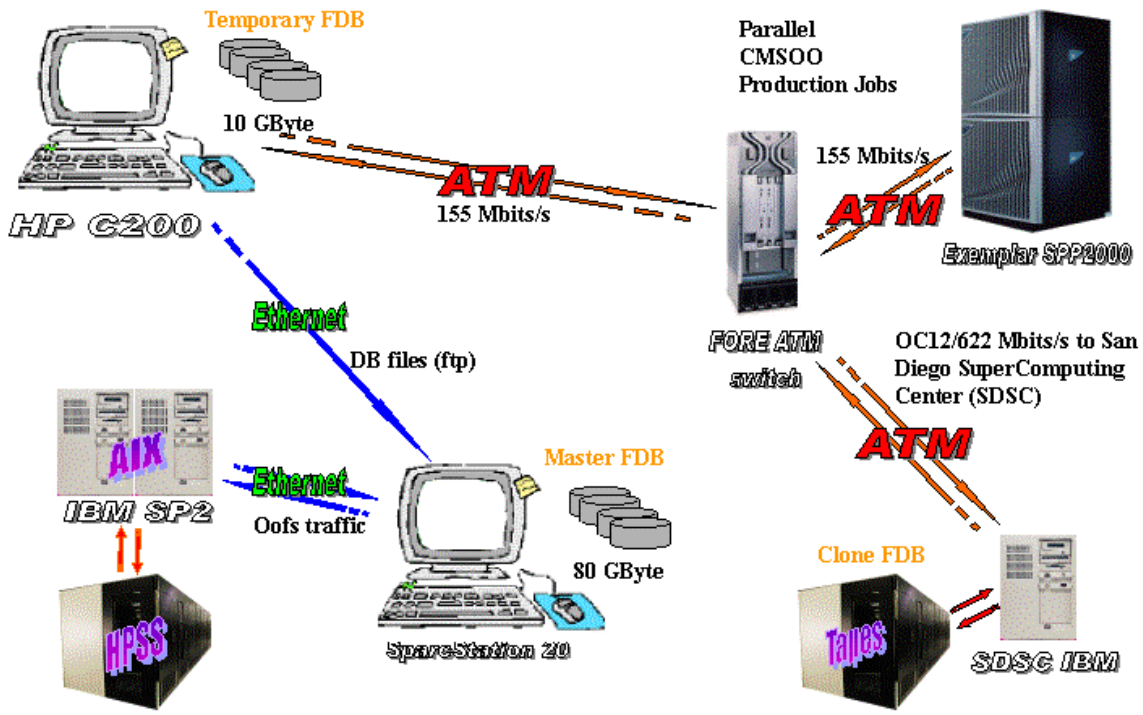


Figure 37: The network and machine configuration for the GIOD WAN tests with SDSC

The route from the SDSC SP2 to the CACR Exemplar is shown below.

1 tigerfish.sdsc.edu (132.249.40.11) 1 ms 1 ms 1 ms

2 medusa.sdsc.edu (132.249.30.10) 1 ms 1 ms 1 ms

3 SDSC-campus-ATM1.calren2.net (198.32.248.62) 2 ms 2 ms 2 ms

4 USC-SDSC.calren2.net (198.32.248.33) 6 ms 5 ms 6 ms

5 UCI-USC.POS.calren2.net (198.32.248.17) 7 ms 7 ms 7 ms

6 UCR-UCI.POS.calren2.net (198.32.248.13) 8 ms 15 ms 8 ms

7 CIT-UCR.POS.calren2.net (198.32.248.9) 10 ms 10 ms 10 ms

8 10.26.254.253 (10.26.254.253) 10 ms 10 ms 10 ms

9 10.26.1.2 (10.26.1.2) 10 ms 10 ms 10 ms

10 SFL-border.dmz.caltech.edu (192.12.19.252) 11 ms 11 ms 11 ms

11 pengine-145.cacr.caltech.edu (131.215.145.250) 11 ms 11 ms 11 ms

12 neptune.caltech.edu (131.215.145.111) 12 ms 11 ms 12 ms

Since the end cable to the San Diego machine is FDDI not ATM, a different command, 'netstat -v FDDI' was used in conjunction with awk to extract the packet count. Roughly 553 bytes belong to each FDDI packet.

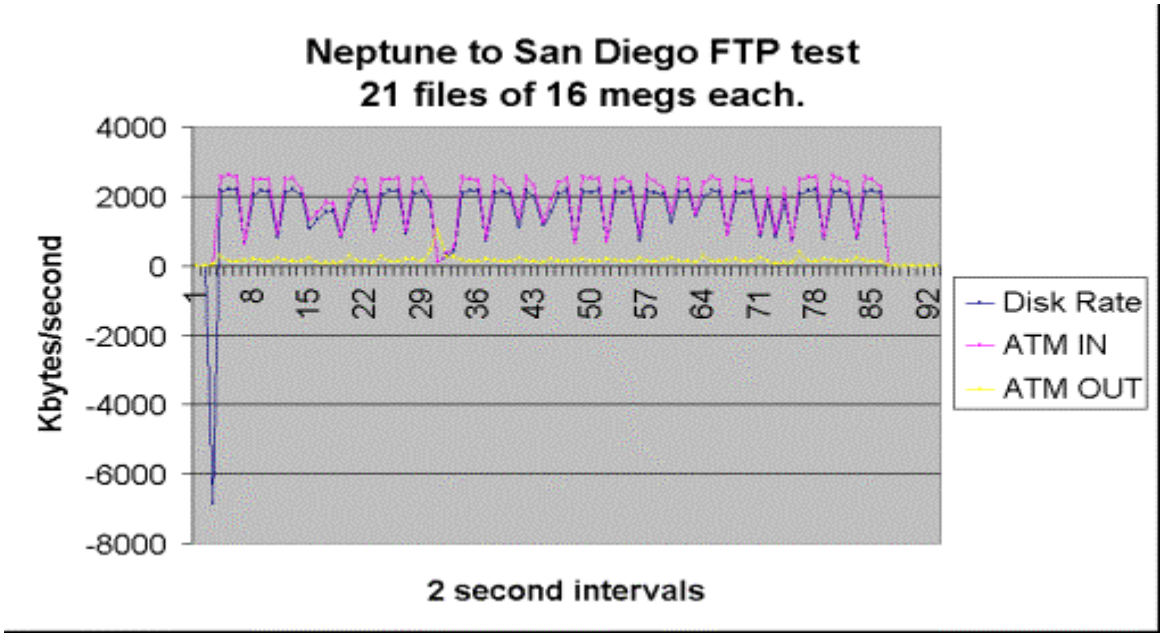


Figure 38: FTP test with SDSC

This graph shows extremely even and ordered behavior. Unlike the ATM, The FDDI line from San Diego is symmetrical and has more even packet arrival times.

More FTP Tests

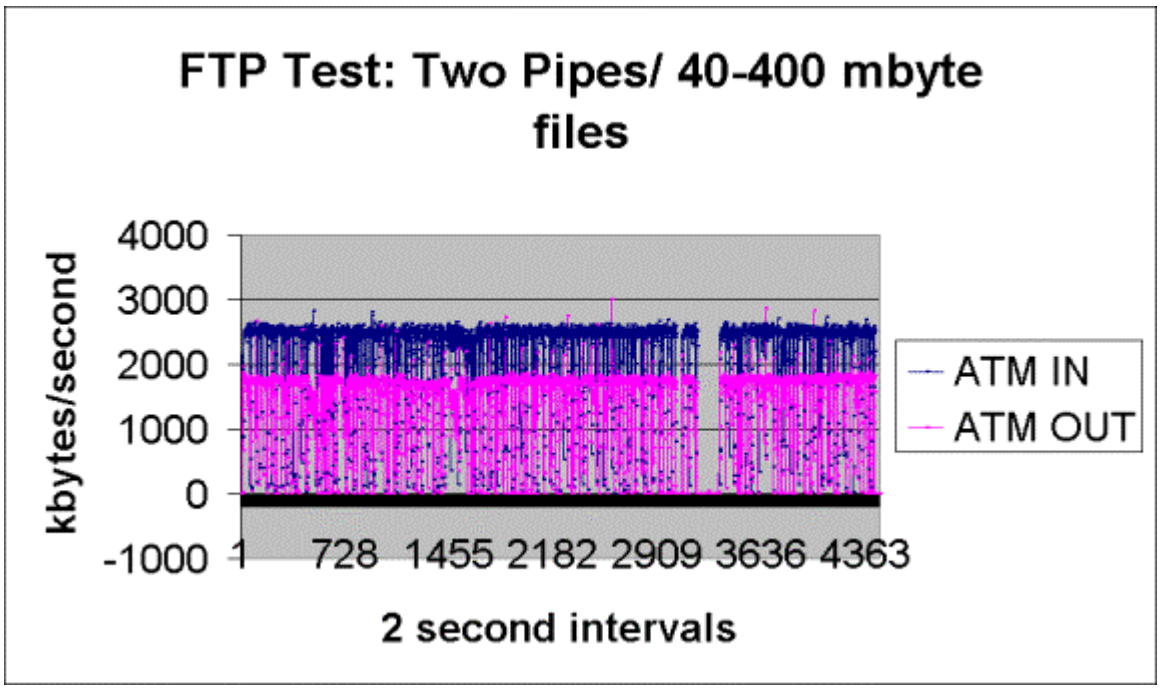
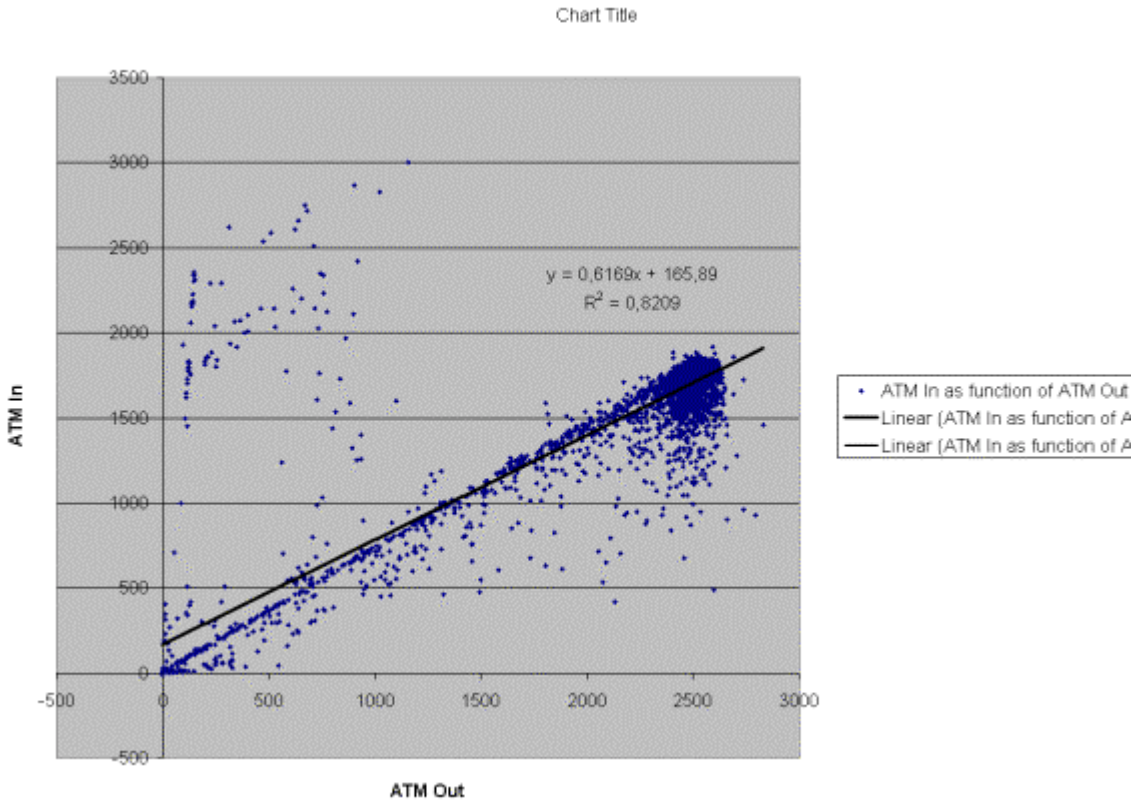


Figure 39: Using two FTP streams simultaneously in an effort to saturate the WAN



The massive FTP test was conducted with two simultaneous pipes. It's interesting to note that the spiking behavior is identical to the same tests with only one pipe.

CMSOO Reconstruction to a database at SDSC

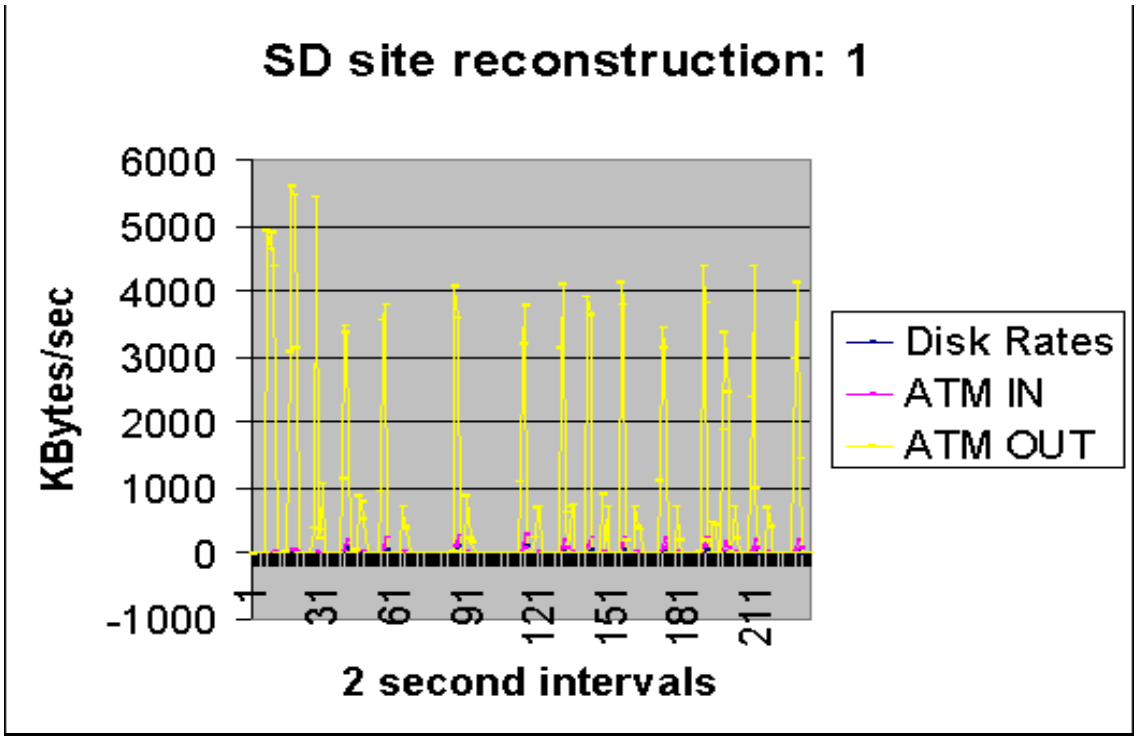


Figure 40: Showing the ATM traffic for CMSOO running on the Exemplar, writing to a database on the SP2 at SDSC.

The single client CMSOO test to San Diego shows excellent network performance (peaking at 5.5 Mbytes/second in the WAN), which is comparable with the performance on the dedicated ATM test bed here at Caltech.

Conclusions from the detailed ATM tests

One of the key elements which can be altered easily is the packet size limit. In broadband networks such as ATM, it is often claimed that using large packets results in greater efficiency. However, during congestion periods, large packets have a higher probability of being lost, which results in a drop in overall throughput. For instance, in a study of the effects of packet size on overall throughput under various network loads throughput degradations were observed for large packet sizes during periods of congestion. In such cases, reducing the packet size **can avoid excessive packet loss and will, in turn, improve throughput performance**. It would be perhaps useful if the packet size could be changed dynamically to adapt to prevailing WAN conditions.

MONARC related work

Tiered distributed system architecture

GIOD maintained very close relations with the MONARC project from the latter's inception. In particular, the modeling work already begun in GIOD (see the sections on SoDA and ModNet below) was continued in MONARC, and further developed with the provision of a more appropriate modeling tool written in Java. The GIOD studies assumed a probable architecture for data distribution at the LHC that is depicted in the schematic below:

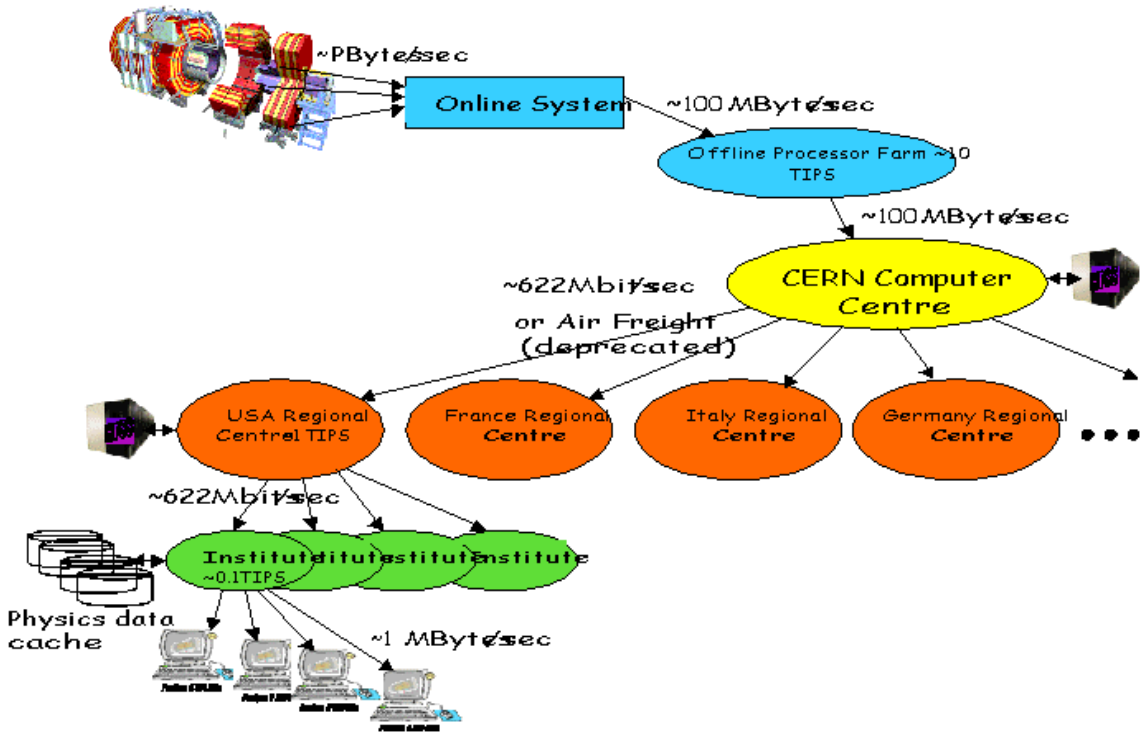


Figure 41: A possible data distribution strategy for one of the LHC Experiments

Modelling the “Average Physicist” using ModNet

In an attempt to understand what the network demands might be to support end user physicists working with the distributed system in 2005, a model was proposed (by Stu Loken) of what a typical physicist might do during the course of a day to impact the network. These tasks are summarized in the following table:

Table 6: Network-related tasks of an average physicist in 2005

Task	Hours Per Day	KBits/sec	Total Mbits
Conferencing	2	512	4,000
"Coffee Room"	0.5	2,000	4,000
Seminar	0.4	1,000	1,600
Remote Sessions	2	256	4,000
Analysis, Including Transfer in background	4	700	10,000
Electronic Notebooks	2	100	800
Papers: 20 Papers and Documents, Including Links	2	100	800
E-Mail: 500 Multimedia Messages	2	50	400
Interactive Virtual Reality	0.5	2,000	4,000
TOTAL MBits Transferred Per Day			30,000
Average Mbps During a Ten Hour Day			0.75

The simple sum of traffic rates gives rise to an average of 0.75 Mbitse/second throughout the working day. The modelling aim is to verify whether the tasks above do indeed give rise to network traffic averaging 0.75 Mbps, and to show the traffic fluctuations over the course of the day. We would also like to model a group of physicists using the above data, and see what the network implications are.

Setting up the Model

To model this user with the ModNet tool, we set up tasks with the same names, which will execute at the required frequency. We choose a work day as being 10 hours long. To model the tasks, we examine what happens each second, when the user is engaged on that task. For example, the "Conferencing" task involves sending and receiving audio/video data. In one second, we say that 512 KBits of data are received, and in the next second 512 KBits of data are sent (this should probably be changed to an 80:20 rule in favour of data reception, but for the moment we keep it simple). The task in ModNet looks like this:

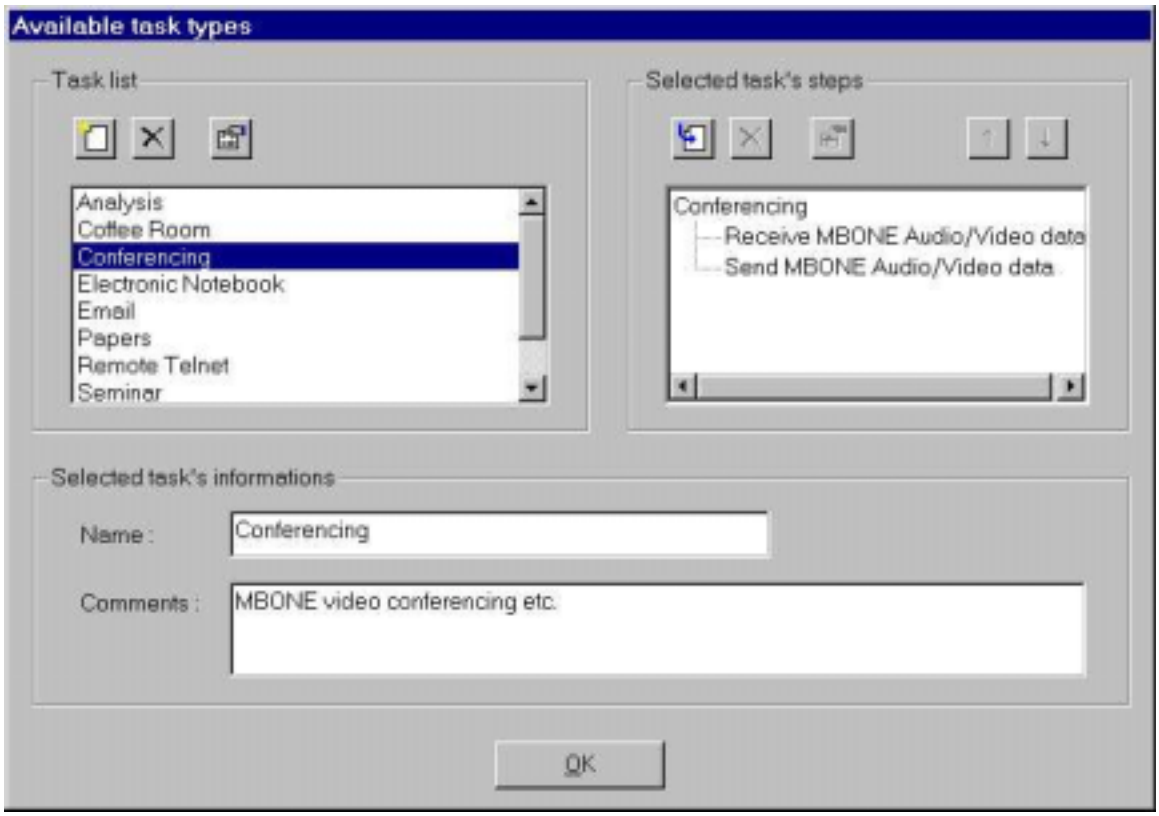


Figure 42: Selecting the task type in ModNet

The two task steps "Receive MBONE Audio/Video data" and "Send MBONE Audio/Video data" are defined in terms of data size and location:

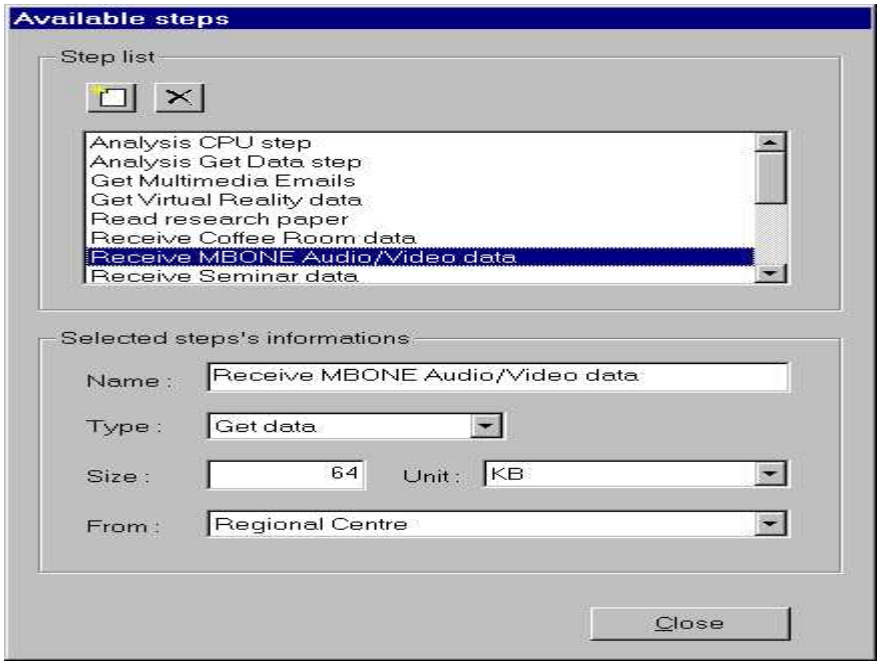


Figure 43: Defining the task steps in ModNet

Note that the data comes from "Regional Centre", and the size of the data is specified in units of KBytes.

The physicist is "Conferencing" for 2 hours every day. There are 7200 seconds in 2 hours, so the above task must be invoked 7200 times over the course of the day. (We are assuming that the physicist is multiplexing between all the different tasks, at the rate specified by Loken's table).

The user works on a single workstation, which is connected to the "Regional Centre" via a network of semi-infinite bandwidth. The completed network model looks like this:

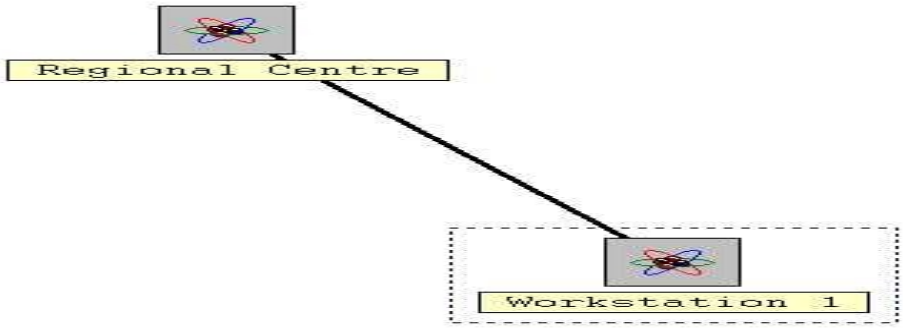


Figure 44: The simple network system for the ModNet simulation

Modelling results

A graph of the network traffic between the physicist's workstation and the "Regional Centre" shows the expected average rate of ~0.72 Mbps. (Note the spikiness of the traffic, peaking at 2.05 Mbps and with a low of 0.09 Mbps).

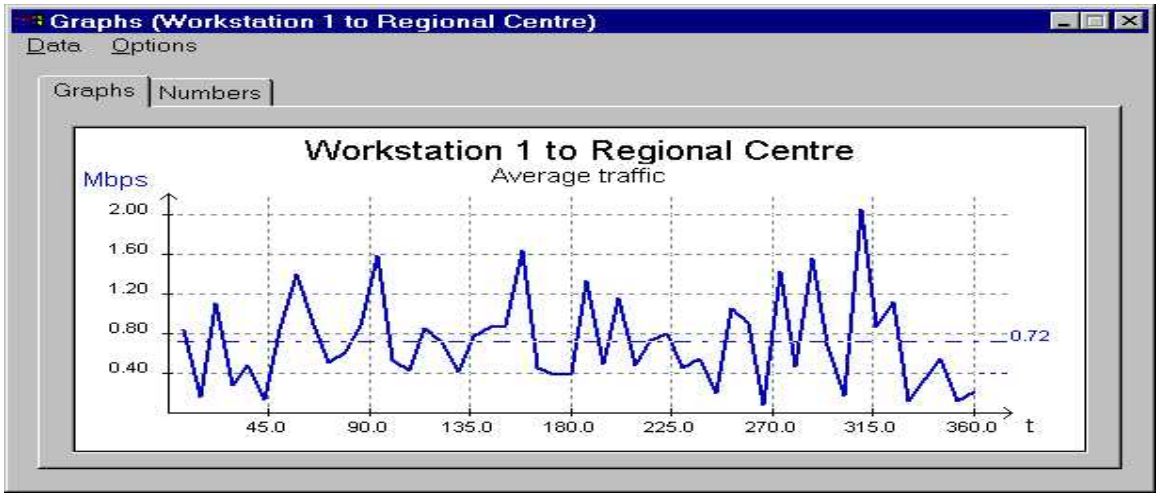


Figure 45: Traffic profile measured in the simulation

Now if we modify the network so that it runs at 1 Mbps, we obtain the following traffic pattern:

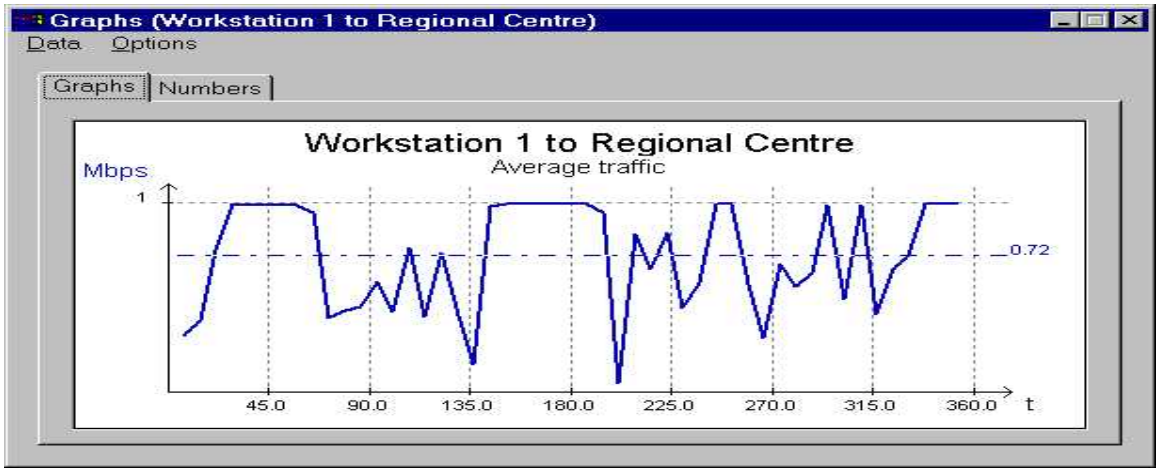


Figure 46: Traffic profile when the network capacity is limited to 1 Mbps

SoDA

The modelling and simulation of distributed computing systems has been investigated in CERN's Physics Data Processing group previously and a framework for the Simulation of Distributed Architectures (SoDA) has been developed. SoDA has been predominantly used for the simulation of local-scale distributed systems with a deterministic workload profile. The present studies should investigate if SoDA is also suited to address particular characteristics of wide area networks such as:

- Hierarchical composition of network constituents: logical and physical topology
- Indirect characterisation of the network load: Tasks - Users - Workgroups – Institutes
- Stochastic modelling of the network load over certain periods of time (working day)

The problem naturally suggests to separate the specification of users (who issue the workload) and network resources (which handle the workload). We will address both aspects separately in the following.

Specification of the Users

The workload is understood as the entirety of data transfer requests that are addressed to a (dedicated) wide area network. The current model understands the total workload as mutual data exchanges among physics institutes, e.g. Caltech and CERN. A particular data exchange results thereby from a wide area utilisation profile of individual physicists. In the case of the given model, such an individual utilisation profile is characterised by 'wide area network tasks of an average physicist in 2005' as proposed by Stu Loken. A task represents a wide area network data transfer / session of an individual physicists related to a certain purpose.

As a preliminary heuristic, the total data transmission requirements of a task of limited duration are characterised either through an upper bandwidth or a volume constraint. The execution of a task is limited to a certain period per working day (start daytime and stop daytime). The execution of the task can be sparsed over several lots per day (lots per day). Each lot accounts for an equal share of a the total requirements. The lots are uniformly distributed over the considered period of a day. If some requirement can not be fulfilled or can be only fulfilled with delay (i.e. time that exceeded stop daytime), the respective task collects statistics on this.

We have explained how tasks serve to model the behaviour of an 'average physicist'. In order to extrapolate the behaviour of multiple users, a workgroup is conceived as a set of users, an institute in turn is understood as a set of workgroups.The total workload issued per institute is thus the sum/overlap of individual workloads induced by workgroups and users.

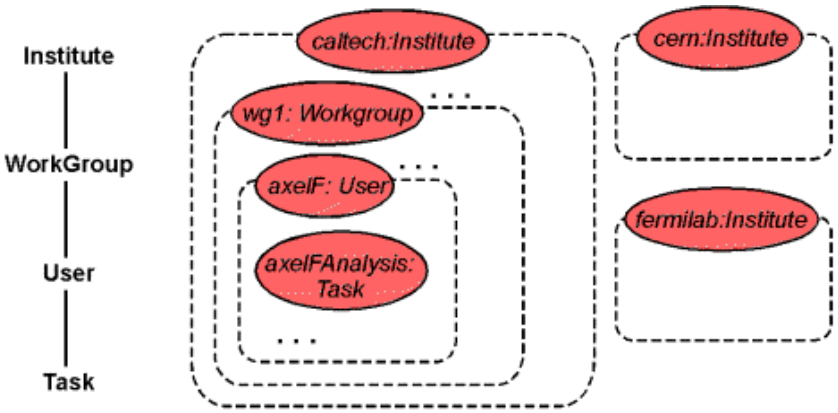


Figure 47: The component model for entities that issue/determine the workload. Components of this category are depicted red.

The structure and properties of entities who issue workloads are assumed to be static during a simulationl and are thus defined in terms of a SoDA component model. The aggregation of tasks to users, users to workgroups and workgroups to institutes is modelled through a component hierarchy as illustrated in figure 1. The hierarchy can be grasped as a composition of behaviour, i.e. the behaviour of an institute shall be composed by the behaviours of its workgroups etc.. Behaviour in this context is understood as the issuing of processes that represent the actual workload. According to the task descriptions in table 1, such behaviour can be characterised by the frequence, the volume of data emission and further parameters. In general, the behaviour of a component parameterised through its attributes. The attributes of a component instance are intialised at creation time through a configuration file. The following configuration file excerpt illustrates for example the intialisation information of the task name 'phy01cernCoffeeRoom':

```
...
[phy01cernCoffeeRoom]
duration                =                0.5              #              [h/day]
bandwidthSend           =             1000.0              #              [kbit/s]
bandwidthReceive        =             1000.0              #              [kbit/s]
requirementsSend        =               -1.0              #              [Mbit]
requirementsReceive     =               -1.0              #              [Mbit]
granularity             =                  1                           #  [#]
startTimeOfDay          =               10.0          #               [local time]
stopTimeOfDay           =               11.0                #           [local time]
...
```

Specification of the Network Resources

Network resources specify all constituents that jointly carry out the requests / workloads. The scenario presented here is a strongly simplified image of reality. It demonstrates nevertheless how SoDA modelling concepts can account for particular structural characteristics of wide area networks. We illustrate complex entities as clouds. In the given model, one of the clouds, namely the Network entity, is refined and an explicit component model is defined for it. The other clouds in the figure are characterised by standard components taken from the SoDA library. These components offer a standard behaviour that can be customised to a certain degree through parameters. These clouds can be refined if further investigation is required, e.g. if a bottleneck is suspected.

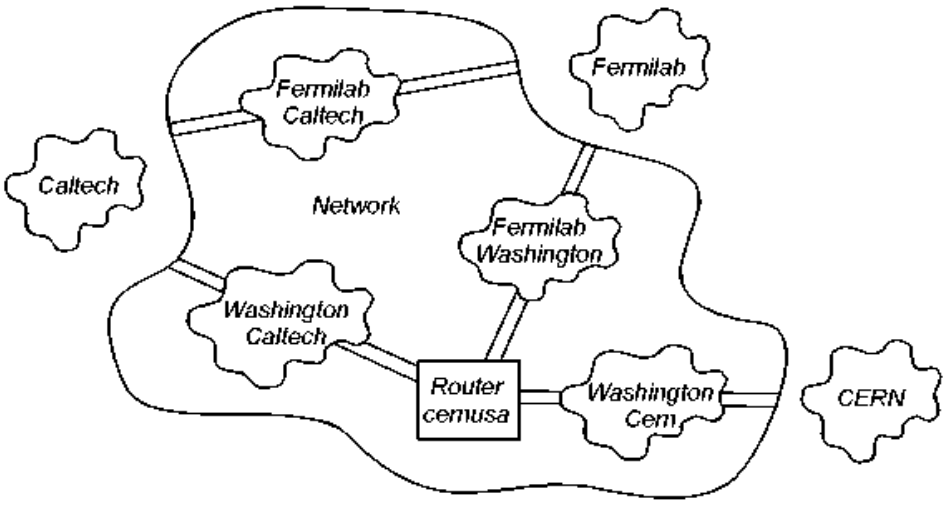


Figure 48: Simplified scenario of a wide area network that connects High-Energy Physics sites

The logical network topology foresees full connectivity among the sites Caltech, CERN and Fermilab. Every logical end-end transfer crosses the Network entity. Within the Network entity, end-end connections are resolved and a physical transmission path is determined according to routing strategies. A physical path corresponds to a series of links (e.g. Washington Cern) and routers (e.g . Router cernusa).

The distinction between logical and physical structure of a wide area network, is represented by a hierarchy of components in the SoDA model. One component class is foreseen for each of the abstractions Network, Router and Link.

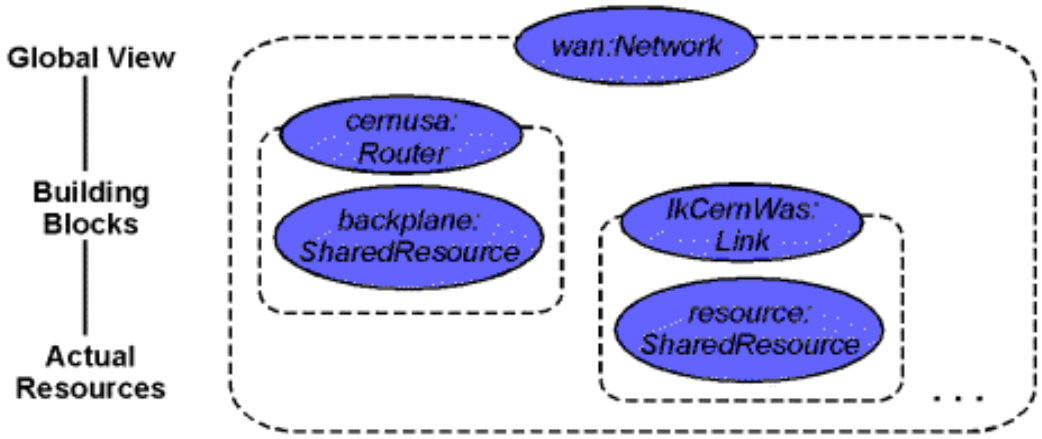


Figure 49: The component model of the wide area network indicates a hierarchical structure.

Components that represent logical elements in the network featuring a high level of abstraction, can be found up in the hierarchy, e.g. the component instance 'wan' of class Network. The behaviour of this high-level component is composed by the behaviour of further sub-components that represent a lower level of abstraction. Theses are of class Router (e.g. instance 'cernusa') and of class Link (e.g. instance 'lkCernWas'). The components of class Router and Link in turn have their behaviour determined through a standard SoDA library component which is instanced from class SharedResource.

Interaction between Users and Network Resources

So far, entities that issue workloads and that handle workloads have been defined in component models. The actual workload as dynamic element is featured by the SoDA modelling concept of processes. The only entities that actually issue processes are those instanced from class Task. It should however be emphasized that the existence and intialisation information of a Task object depends on the hierarchy of Users - Workgroups and Institutes as discussed in the previous section. Thus also those objects of class User, Workgroup and Institute are in some sense 'indirectly' characterising the workload.

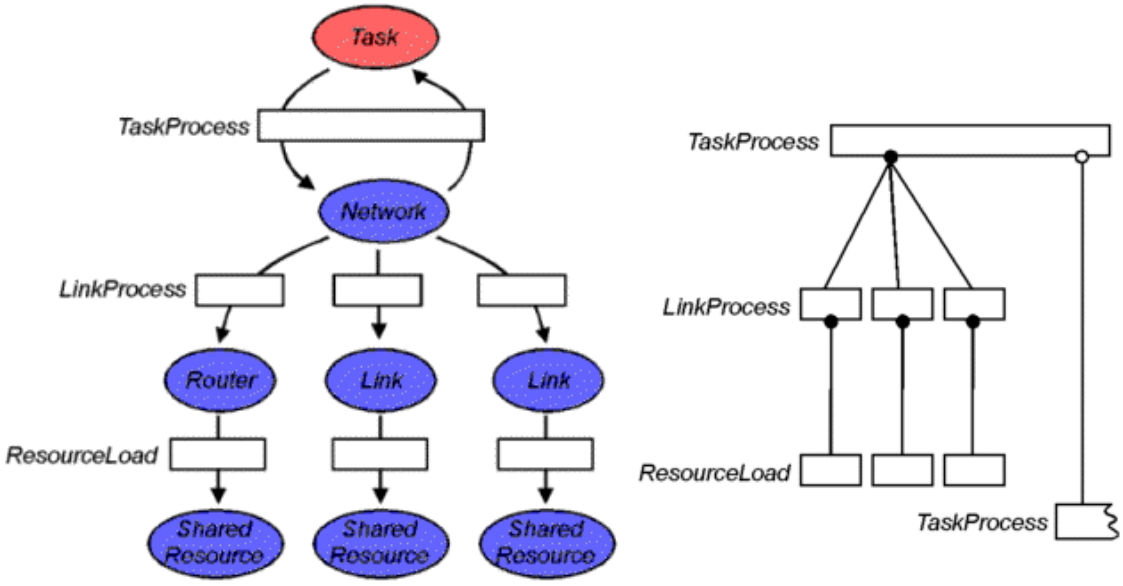


Figure 50: The interaction between components that

issue workload (red) and components the handle workload (blue) is modelled by processes.

Tasks do thus issue processes of type TaskProcess in certain intervals in simulation time. The length of the interval is determined by a heuristic. The heuristic calculates the value from the attributes 'lots per day', 'start daytime' and 'end daytime'. In addition, the heuristic foresees that no TaskProcess may be created until the previous instance has finished execution (see process model at the right hand side of figure 3). The process model illustrates that a TaskProcess process is split into several sub-processes that execute concurrently on all components that belong to a certain network path. This heuristics of concurrent execution is feasible if the TaskProcess itself represents a streamed data transfer. For small data drops of data issued by an interactive session, a different heuristic may be necessary.

Simulation and Analysis

The behaviour of a SoDA model can be observed either during its evaluation or after a simulation run. The following examples of visualised result data illustrate either way.

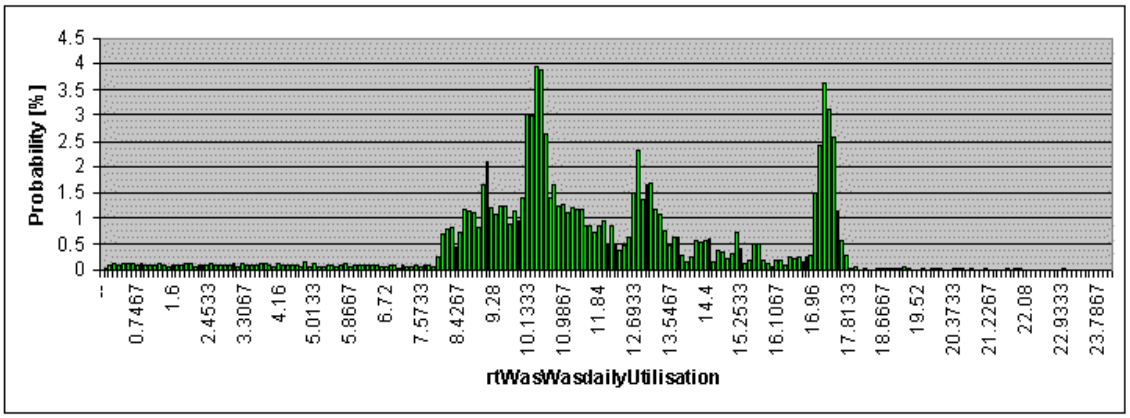


Figure 51: The utilisation of a particular network link over 24 hours.

In the above Figure one clearly recognises certain prominent, i.e. bandwidth intensive, tasks such as the Coffee Hour between 10 and 10:30 and the Virtual Reality session around 17:30 in the afternoon. The background data transfers during the night result from the Background Analysis task that was characterised by a 24 hours duration, small bandwidth requirements and a fine granularity. The averaged utilisation over the 24 hours period is in exact concordance with figures that can be easily calculated from the task descriptions. The ordinate in the figure indicates a probability [%], which results from the algorithm that generated the histogram after the simulation run. The y-scale can be easily (linearly) converted to utilisation figures or absolute bandwidth consumption.

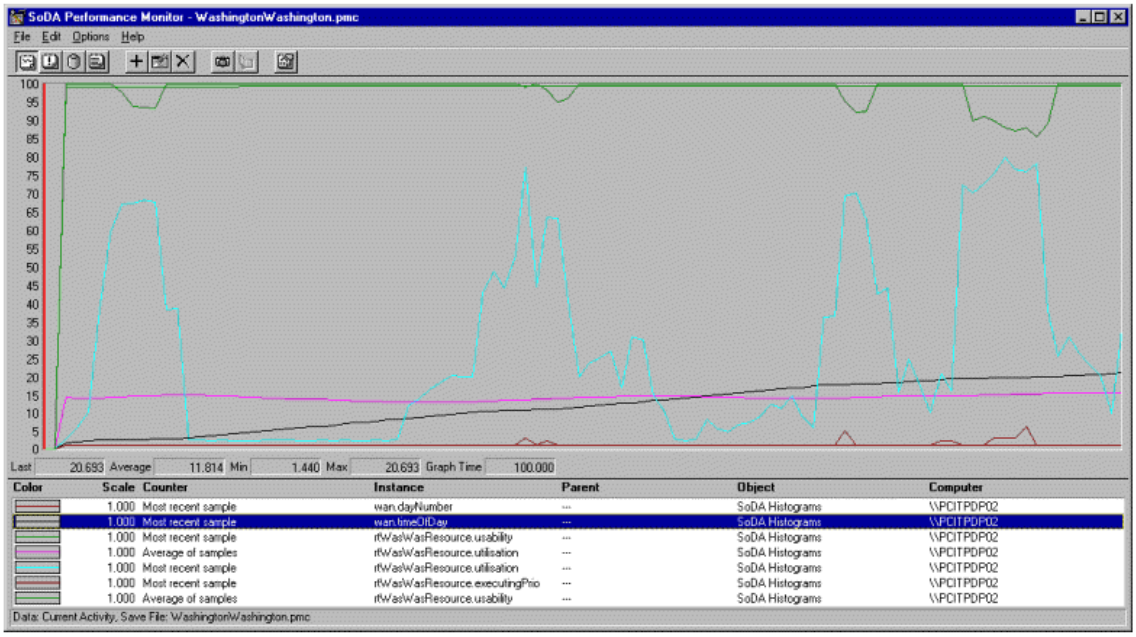


Figure 52: The behaviour of a Router component as it is observed during the simulation of the model with the SoDA Performance Monitor.

The displayed segment illustrates the values of various observable attributes of the component during a working day such that the leftmost ticks of the graph represent the status at midnight. The straight black line illustrates the daytime which increases steadily up to 24 hours. In the simple model, a router is conceived as a backplane with limited capacity where all data transmissions pass through. The utilisation of this backplane is determined in intervals of 192 simulated seconds. The value is depicted as light blue graph. The capacity of the backplane (represented by a standard SoDA component of type SharedResource) has been laid out to a reasonable value such that utilisations up to 75% could be observed for a model with 4 physicists in working in two different time zones. The long-term average utilisation is indicated by the purple line. The brown line indicates the number of concurrent streams that induce traffic on the router. The green line indicates the absolute usability of the resource. As a heuristic, the router component decreases its bandwidth capabilities in case of sustained high utilisation. This heuristic should meet observations of packet loss and flow control mechanisms in transport protocols. It was used experimentally here and is subject to change.

Future Work

During the exercise, it turned out that modelling the wide area scenario at hand is mainly determined by two aspects:

Structural Aspect: Structure in this context addresses the transfer, allotment and distribution of any kind of workload from the issueing component of the workload to the handling component. In the current model, the complex lifecycle of the workload is characterised in terms of SoDA processes that 'flow' from a hierarchy of load issueing components through a hierarchy of load handling components (see figure 7)

Qualitative Aspect: The qualitative aspect of a model addresses how workload is created and handled. This is determined by heuristics that determine the behaviour of load issuing and load-handling components.

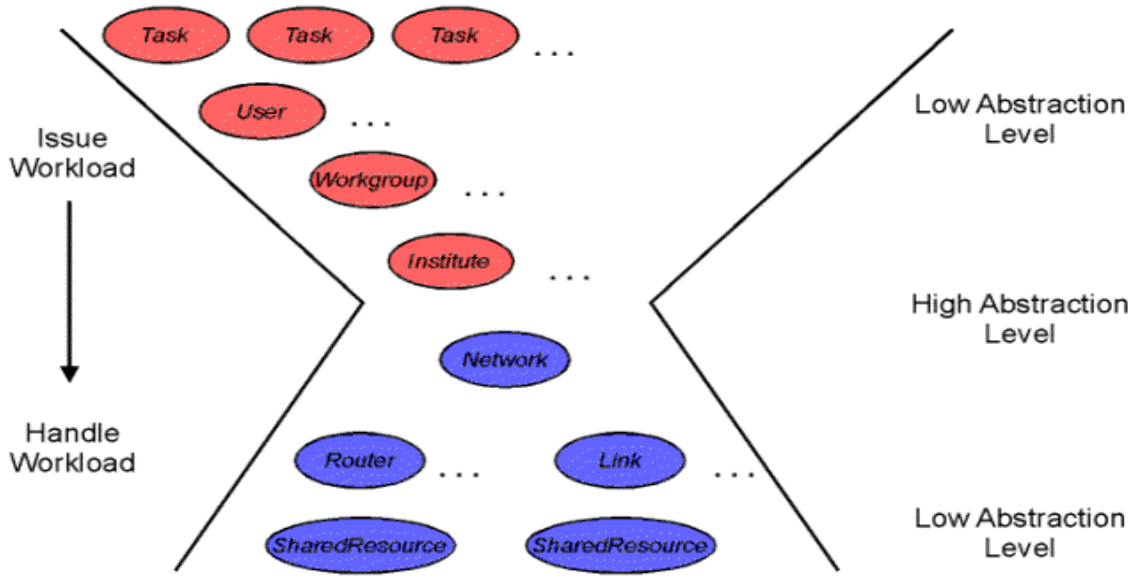


Figure 53: The structural aspect of a model: many tasks are effected (through workload processes) on many resources concurrently

It is demonstrated that the SoDA modelling approach can efficiently represent the structural aspect of modelling. A strategy for a continuation of this work would thus rather focus on the development of the qualitative aspect, i.e. the definition of heuristics.

The applied heuristics for tasks featuring application network protocol and users (being a collection of tasks that are pursued concurrently), are insufficient to represent and understand today's observed network behaviour with the developed model. In literature, different approaches have been taken to match wide area traffic with formal methods. These approaches range from a strict empiric proceeding to a mathematically sound formalisation with different flavours of distributions. Initial approaches have been started to characterise the behaviour of a user who is confronted with a finite 'resource' with concepts from game theory.

In addition to heuristics for load issueing elements, it is feasible to implement non-trivial heuristics for workload handling components such as Router components. The observation of the Router component with the SoDA Performance Monitor already suggested a deterioration of its capability in case of lasting high utilisation. It should however be emphasised that the implementation of complex heuristics on both sides - the load issueing and the load handling side - may 'overcook' the actual problem. This may render the behaviour of the model 'incomprehensible'. Finally, traces of network traffic and their empirical analysis may already take a deterioration of certain load handling component into account. We thus suggest to develop heuristics rather on the origin side of the workload, i.e. heuristics for the behaviour of users and application protocols.

Summary of the SoDA work

The GIOD work has resulted in the construction of a large set of fully simulated events, and these have been used to create a large OO database. The Project has demonstrated the creation of large database federations, an activity that has included practical experience with a variety of associated problems. We have developed prototype reconstruction and analysis codes that work with persistent objects in the database. We have deployed facilities and database federations as useful testbeds for Computing Model studies in the MONARC Project.

The project has proved to be an excellent vehicle for raising awareness of the computing challenges of the next generation of particle physics experiments, both within the HEP community, and outside in other scientific communities.

The JavaCMS event viewer was a featured demonstration at the Internet-2 Fall Meeting 1998 in San Francisco. At this event we used an HP Kayak PC with FX4 graphics card as the demonstration platform, a machine loaned to us by Hewlett Packard. At the SuperComputing '98 conference, the event viewer was demonstrated at the HP, iGrid, NCSA and CACR stands, on two HP Kayak machines dedicated by Hewlett Packard. At the Internet2 Distributed Storage Infrastructure workshop at Chapel Hill, NC in March 1999, the GIOD Project was a featured potential application. GIOD will be present at the EDUCAUSE conference in October 1999.

This has strengthened our relationship with Internet-2, and (together with the VRVS work) is paving the way for CERN to become a fully-fledged Internet-2 member later in 1999.

References

1. "Scalability to Hundreds of Clients in HEP Object Databases", Koen Holtman, Julian Bunn, Proc. of CHEP '98, Chicago, USA

2. "Status Report from the Caltech/CERN/HP "GIOD" Joint Project - Globally Interconnected Object Databases", Julian Bunn, Harvey Newman, and Rick Wilkinson, Proc. of CHEP '98, Chicago, USA

3. "GIOD - Globally Interconnected Object Databases", Julian Bunn and Harvey Newman, CACR Annual Report 1998

4. "Global Initiatives Challenge Traditional Data Management Tools", Internet-2 Press Release, September 1998

5. "Caltech HP Exemplar Supports Test of Data Analysis from CERN Large Hadron Collider", Julian Bunn and Tina Mihaly, NPACI "Online" Article, April 1998

6. "Data Topics", Electronic News article, December 1, 1997

7. "CERN, Caltech and HP open scientific datacenter", HPCN News Article, November 1997

8. "Large-scale Scientific Data-analysis Center Will Address Future Computing and Data-handling Challenges", Hewlett Packard Press Release, November 1997

9. Christoph von Praun: *Modelling and Simulation of Wide Area Data Communications*. A talk given at the CMS Computing Steering Board on 19/06/98.

10. *SoDA Web pages* at http://wwwinfo.cern.ch/~praun/soda/Welcome.html.

GIOD Project Participants

James Amundson, FNAL
Eva Arderiu-Ribera,CERN/IT/RD45
Greg Astfalk, Hewlett Packard
Josh Bao, Caltech/HEP
Saskya Byerly, Caltech/HEP
Julian Bunn, CERN/IT and Caltech
Koen Holtman, CERN/EP
Vincenzo Innocente, CERN/ECP
Paul Messina, Caltech/CACR
Shahzad Muzaffer, FNAL
Harvey Newman, Caltech/HEP

James Patton, Caltech/CACR
Ruth Pordes, FNAL
Sergey Shevchenko, Caltech/HEP
Christoph von Praun, CERN/IT/PDP
Rick Wilkinson, Caltech/HEP
Roy Williams, Caltech/CACR

Ackowledgements

We acknowledge and thank the following individuals:

Shane Davison, Objectivity (for technical support)
Dirk Duellman, CERN/IT (for general help with Objectivity)
Manuel Delfino, CERN/IT (for CERN/IT support as Division Leader)
Juergen May, CERN/DG (for initial CERN/IT support as Division Leader)
Reagan Moore, SDSC (for the SDSC WAN test accounts)
Les Robertson, CERN/IT (for crucial technical and managerial support throughout)
Tom Sherwin, SDSC (for implementing the SDSC WAN test configuration)
Jamie Shiers, CERN/IT (for general help with Objectivity and RD45 liaison)

Publications and Press

"Scalability to Hundreds of Clients in HEP Object Databases", Koen Holtman, Julian Bunn, Proc. of CHEP '98, Chicago, USA

"Status Report from the Caltech/CERN/HP "GIOD" Joint Project - Globally Interconnected Object Databases", Julian Bunn, Harvey Newman, and Rick Wilkinson, Proc. of CHEP '98, Chicago, USA

"GIOD – Globally Interconnected Object Databases", Julian Bunn and Harvey Newman, CACR Annual Report 1998

"Global Initiatives Challenge Traditional Data Management Tools", Internet-2 Press Release, September 1998

"Caltech HP Exemplar Supports Test of Data Analysis from CERN Large Hadron Collider", Julian Bunn and Tina Mihaly, NPACI "Online" Article, April 1998

"Data Topics", Electronic News article, December 1, 1997

"CERN, Caltech and HP open scientific datacenter", HPCN News Article, November 1997

"Large-scale Scientific Data-analysis Center Will Address Future Computing and Data-handling Challenges", Hewlett Packard Press Release, November 1997