CDF/DOC/COMP_UPG/PUBLIC/4766
CDF$PUB:CDF4766_DH_proto.ps
Version 1.2 January 9, 1999

The DH Resource Management Prototype (CDF Run II Data Handling)

W.Brown, E.Buckley-Geer, S.Lammel, T.Watts, E.Wicklund

January 9, 1998

Abstract

For the data handling system of CDF for Run II, we describe a prototype of the software architecture and resource management methods that were proposed in CDF Note 4759. Results are described from a simulation of the system, focussing in particular on the disk inventory software and batch queues expected to manage cpu, disk, and tape resources. This is a feasibility test for this disk inventory software and for this method of resource management using batch queues.

Contents

1 Introduction

This document concerns the analysis stage of data handling, and the associated tasks of manipulating data. Processing of raw data in farms, and the storing of raw data at collision time, are discussed elsewhere.

We have proposed, in a previous document [1], that the allocation of disk, tape, and cpu resources in the data handling system among the many data analysis jobs generated by the experimenters will be by batch queues. Two classes of batch queues were postulated, one queue for jobs analyzing data by reading it from or writing it to managed (staging) disk, and the other class of queue to manage the staging of data from tape to disk and vice versa. In this document, we report a simulation prototype of this system and discuss a set of simulated data analysis jobs to find out if our batch queue scheme can efficiently allocate cpu, disk space, and tape access, given the expected demand in Run II.

The various simulation jobs used for the test involve several mixtures ("scenarios") of different types of analysis jobs running on a scaled down data handling (DH) computer. The limited extent of the prototype DH computer enabled tests which simulated overload conditions as well as study of the competition between different types of analysis jobs.

Important features of the resource management scheme evolved during the prototype tests, as can be seen below, and the notes about future versions of the scheme, especially the Disk Inventory Manager (DIM), have been collected in the relevant sections below.

2 Features of the Data Handling Software

User jobs will read from or write to disk; there will be no tape access by analysis jobs using ROOT I/O (the CDF data format). For data on tape therefore, staging software will transfer files between disk and tape. Usually the transfer will involve a complete tape or "fileset" (the concept of the fileset allows for tape partitions).

Management of scarce resources will be needed and a scheme has been proposed [1]. The resources to be managed are cpu cycles, disk space for data, and tape drives. Disk I/O bandwidth and size of the tape library (both in the automatic mounting robot and on shelf) are not expected to need dynamic management.

Normal batch queue priorities will manage cpu time and allocation of tape drives. Allocation of disk space will be governed mainly by the staging software, i.e. the Disk Inventory Manager software and associated databases and daemon(s).

The user's view of the software organisation is shown in Figure 1 (taken from [1]). The user submits a data analysis job to an analysis job batch queue ("CPU queues" in the diagram). The AC++ I/O module in the user's job translates the dataset name into a list of filesets using the CDF Data Catalog, gives the list to the Disk Inventory Manager who returns with the list of filesets on disk, if any. Staging jobs are submitted to the I/O Queue to fetch filesets not on disk from tape. Meanwhile, the user job processes filesets on disk. The writing of datasets was not tested in the prototype, only reading.

3 Prototype and its Simulation

The prototype consists of:

- A set of basic batch queues of both classes, cpu and I/O, set up in the LSF product [2] (which is a candidate for final adoption).
- Staging software with all basic functions but not running in final multi-cpu architural form (Disk Inventory Manager).
- Simulated analysis jobs which read dummy events from files but do not process real data.
- A set of big and small dummy datasets (not real data).
- Basic CDF Data Catalog software with a catalog containing entries for the datasets and files of this simulation.

The purpose of the simulation is to test ideas and evaluate how analysis jobs interact in a resource limited environment. The tests include an evaluation of the features of the staging software and of the batch queues.

4 Scaled Down DH Computing System

The limited environment for the prototype is:

- Single cpu workstation b0ib04.fnal.gov (IBM, R6000, 63 MHz)
- Staging disk of size 9 GB

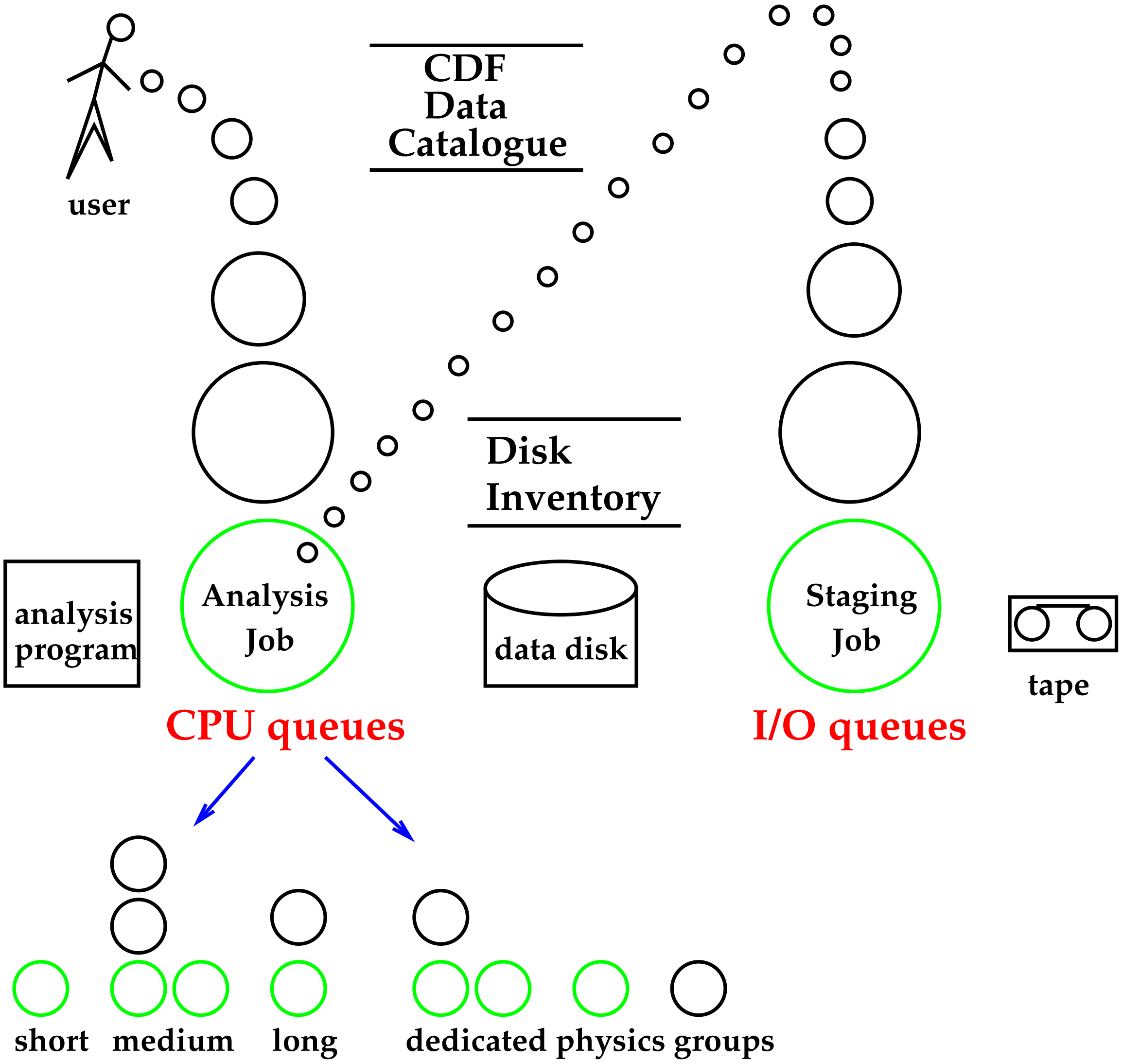

Figure 1: CDF Run II: User's view of Data Handling System

- Two simulated tape drives
- Four small datasets of 1 GB, and four large of 10 GB
- Two cpu queues, short and long, with 4 execution slots each
- Analysis jobs with variable (random) cpu time
- One real tape drive in the Emass robot archive

Discussion:

A convenient workstation for this evaluation is b0ib04 which has one cpu, a small disk for staging data, 1 tape drive, and another disk which could be used to simulate tape drives.

Fake datasets of size 1GB (small) and 10 GB (large) will be used, with files of size 0.1 GB, collected into filesets of 0.5 GB. Four different small datasets and four large have been created (SHORT_A, SHORT_B,... LONG_D). Analysis jobs can then select among this variety of datasets. Note that the small datasets are about 10% of the available staging disk size, and the large will not fit completely on the staging disk. This gives a reasonable approximation to the relative sizes of real datasets (staging disk 20 TB, medium datasets of 2 TB, and very large of 20 TB) but allows for quicker response times during simulation. The datasets contain dummy data.

Tape storage was simulated by another disk drive which helps speed debugging. The number of simulated tape drives is controlled by the number of execution slots in the batch queue "i_o_queue" where staging jobs are submitted. This use of simulated tape drives allows for a greater range of scenarios (see below).

Batch queues have been created using the LSF package, partly in order to evaluate the package. There were two cpu queues, for small and large dataset jobs, with lower priority for large dataset jobs; each cpu queue was given 4 execution slots. There was one I/O queue of higher priority for staging filesets from simulated or real tape to disk; this I/O queue was given 2 execution slots to simulate 2 simulated tape drives.

The disk inventory manager software had all basic functions, as did the CDF Datafile Catalog software.

The AC++ input module portion of analysis jobs was coded as unix scripts for the simulation. The cpu time for a job was simulated by loops which absorbed real cpu time. The parameter controlling cpu time had a range equivalent to a ratio 0.1 to 5 times disk reading speed, approximately.

5 Prototype Disk Inventory Manager (DIM)

The primary element to be evaluated in the prototype is the Disk Inventory Manager. This section mixes information about the full DIM, the prototype implementation, and comments about the full implementation as learned during prototyping.

5.1 General Requirements on DIM

The general requirements on the full DIM are:

- DIM manages (1) the inventory of CDF data (dynamic & static) on spooling disk (local & global), and (2) the spooling of CDF data from tape archive to disk and vice versa.
- DIM receives requests for data from user jobs and reports back the disk location(s) of the data. Vice versa for requests for writing data.
- DIM initiates the transfer of data between tape archive and disk when necessary.
- For the transfer between tape and disk, DIM uses a separate package of tools to interface with tape drives and the robotic archive.
- DIM provides management tools to monitor use of the system and use of data.
- DIM provides user tools for physics group administrators to manage their data.
- DIM provides management tools to debug and improve the system.
- DIM will operate in a heterogeneous unix cluster with multiple users.

5.2 Prototype Implementation of DIM

For the prototype, the DIM was implemented as an executable image invoked with command line arguments among which was a specification of which DIM function was to be invoked. Functions to do with management and user tools were only rudimentary in the prototype, and it was designed to operate in a single workstation environment, not in a cluster. However, an essential element of the prototype was the interaction of multiple users. The prototype executable image returned information as text to its standard output, and this was analysed by the calling process. For the full DH system, a client/server architecture is planned.

5.3 DIM Functionality

A list of functions of the Disk Inventory Manager can be obtained by invoking the stager executable with command line parameter -h, viz. the stage -h command. Here is the output from that command:

```
Stager System, version v0.1.3, usage:
 stage reserve  [-h] [-lnw] tape
                Reserve a fileset for use.
                -h : online help
 stage release  [-h] tape
                Release a fileset that was previously reserved.
                -h : online help
 stage list     [-h] [-s [fileset]] [-f [file]] [-F [filesystem]] [-S]
                List known filesets, files, active filesystems, or parameters
                of the stager system.
                -h : online help
 stage launch   [-h] [input or output] tape
                Launch a stage input or output job.
```

```
                 -h : online help
 stage input     [-h] fileset
                 Stage a fileset from tape to disk.
                 -h : online help
 stage output    [-h] fileset
                 Stage a fileset from disk to tape.
                 -h : online help
 stage allocate [-h] [-s size] fileset
                 Allocate space for a new fileset.
                 -h : online help
 stage update    [-h] [-C] [-s size] -f fileset [file [...]]
                 Update fileset information.
                 -h : online help
 stage help
                 Prints this help message.
Type "stage <subcommand> -h" for more information where <subcommand> is
one of reserve, release, list, launch, input, output, allocate, or update.
```

In the rest of this section, stager functions are capitalized to help the reader; the command line functions are lower case in actual use.

The RESERVE command operated in a mode where all filesets in the list given on the command line were reserved if on disk already. For those filesets not on disk, a stage LAUNCH command was given the list of the missing filesets and issued stage INPUT commands, one per tape volume grouping together filesets on the same tape volume. As filesets appeared on disk, they were reserved by the stage RESERVE process which waited on their appearance. Note that the RESERVE command line list was the complete fileset list for the dataset, so that optimum grouping of filesets by volume could occur, and the batch queue job list could show the total future demand for tape to disk transfers (see below for such a list). If a substantial number of filesets were already on disk, then an unwanted feature in the prototype was that reservations could get too far ahead of the processing rate of a user job; this feature will be removed in the full DIM in order to conserve disk space.

In the prototype, the LAUNCH command submits a stage INPUT command as a batch job in the LSF queue i_o_queue. The choice to use a batch queue, how to submit the job, and to which queue, is controlled by the DIM configuration file (see diagrams in next section), and is not implemented in code (important for non-Fermilab usage).

The stage INPUT command used the stage ALLOCATE function to locate space to put filesets read from tape. If there was not room for new filesets on disk (the usual case since filesets on disk are not deleted until space is needed to fetch another from tape), stage ALLOCATE looked for filesets to delete which: 1) had no reservations, 2) had been a long time unused, 3) had small frequency of past use. The amount of space needed for a fileset was read from the partition data on tape. Retry of failed tape reads was done in the stage INPUT command, but there was not propagation of errors to the user job I/O module in this prototype.

The stage INPUT command in the prototype was normally used to read dummy datasets from another disk, thus simulating tape reading. It could also read datasets which were located on tape in the EMASS robot and occasional scenario trials, see below, were executed to test his feature. The DIM configuration file again was used to control the details of access to tapes and other disks (again important for non-Fermilab usage).

The stage OUTPUT command was not implemented for the prototype.

The stage LIST command was used to find which filesets were on disk. Here is the output from a stage LIST -s command:

```
Known Filesets:
 DMY000.0     (   0.5 GBytes,  5 files) cached on disk
 DMY000.1     (   0.5 GBytes,  5 files) cached on disk
 DMY001.0     (   0.5 GBytes,  5 files) static copy on disk
 DMY003.0     (   0.5 GBytes,  5 files) cached on disk
 DMY003.1     (   0.5 GBytes,  5 files) cached on disk
 DMY017.0     (   0.5 GBytes,  5 files) cached on disk
 DMY017.1     (   0.5 GBytes,  5 files) cached on disk
 DMY018.0     (   0.5 GBytes,  5 files) cached on disk
 DMY020.1     (   0.5 GBytes,  5 files) cached on disk
 DMY021.0     (   0.5 GBytes,  5 files) cached on disk
 DMY021.1     (   0.5 GBytes,  5 files) cached on disk
 DMY029.1     (   0.5 GBytes,  5 files) cached on disk
 DMY038.0     (   0.5 GBytes,  5 files) cached on disk
 DMY046.0     (   0.5 GBytes,  5 files) cached on disk
 DMY046.1     (   0.5 GBytes,  5 files) cached on disk
 DMY047.0     (   0.5 GBytes,  5 files) cached on disk
 DMY047.1     (   0.5 GBytes,  5 files) cached on disk
```

A function missing from the prototype DIM was the ability to list reservations. A script was written to read reservations from the DIM use files (see next section). Here is the output from that script:

```
Filesets   Reservations
DMY000.0
DMY000.1
DMY001.0
DMY003.0
DMY003.1
DMY017.0
DMY017.1
DMY018.0
DMY020.1   dhproto
DMY021.0   dhproto
DMY021.1   dhproto
DMY029.1   dhproto
DMY038.0
DMY046.0   watts  dhproto
```

```
DMY046.1   watts  dhproto
DMY047.0   watts  dhproto
DMY047.1   watts  dhproto
```

Other missing functions from the prototype DIM were connected with cleanup after crashes (likely in user code). This was done manually with some use of scripts.

The list of jobs generated by a single user process reading a dataset is shown, in the prototype, by the LSF batch command bjobs. Below is the output from that command. The list includes the original user analysis job as well as the stage input jobs, some of which are suspended until needed ("PSUSP").

```
JOBID USER     STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2993  watts    RUN   i_o_queue  emass2      emass2      * DMY040.1 Jan  1 16:30
2992  watts    RUN   long       emass2      emass2      *LONG_D 26 Jan  1 16:30
2994  watts    PSUSP i_o_queue  emass2                  * DMY041.1 Jan  1 16:30
2995  watts    PSUSP i_o_queue  emass2                  * DMY042.1 Jan  1 16:30
2996  watts    PSUSP i_o_queue  emass2                  * DMY043.1 Jan  1 16:30
2997  watts    PSUSP i_o_queue  emass2                  * DMY044.1 Jan  1 16:30
2998  watts    PSUSP i_o_queue  emass2                  * DMY045.1 Jan  1 16:30
2999  watts    PSUSP i_o_queue  emass2                  * DMY048.1 Jan  1 16:30
3000  watts    PSUSP i_o_queue  emass2                  * DMY049.1 Jan  1 16:30
```

5.4 Message Paths among Elements of Resource Management System

Components of DIM are shown in Figure 2 labelled "DIM". The component "inv" kept track of which filesets were on or being staged to disk; it was implemented in the prototype as an individual ascii file for each fileset. Similarly for the "use" component which kept track of reservations for filesets. The whole Figure shows the complete resource management scheme.

The sequence of messages using these paths (in Figure 2) during the execution of a user job reading a dataset is shown in the Section 6.1.

The baseline hardware for the DH system, discussed in another cdf note [3], is planned to be a cluster of multi-processor computers ("SMP"'s). There will be both local disk on each SMP as well as disk shared between all SMP's. The shared disks will be read-only in the current plan. A preliminary plan for the resource management scheme in this baseline hardware system is shown in Figure 3. An extra inv component, common to all SMP's, is postulated for the shared disks, and a component for keeping track of DIM statistics and operations is added.

5.5 Administration and User Tools in DIM; Missing Functions

There were few administration and user tools in the prototype (exceptions have been mentioned above). Some thought was given to such tools, however, and here are some brief notes from a talk:

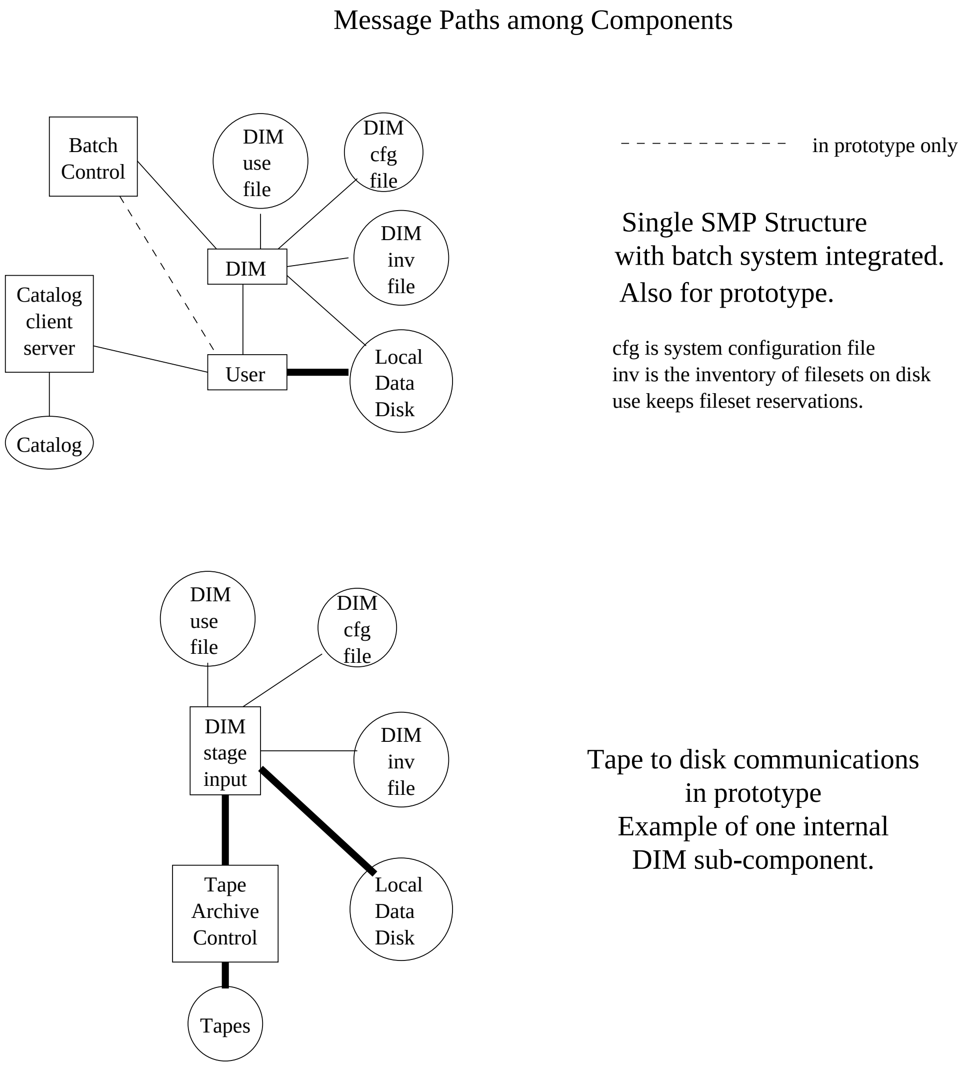

Figure 2: CDF Run II: Components of Resource Management Scheme

Message Paths among Components for Baseline System (Draft)

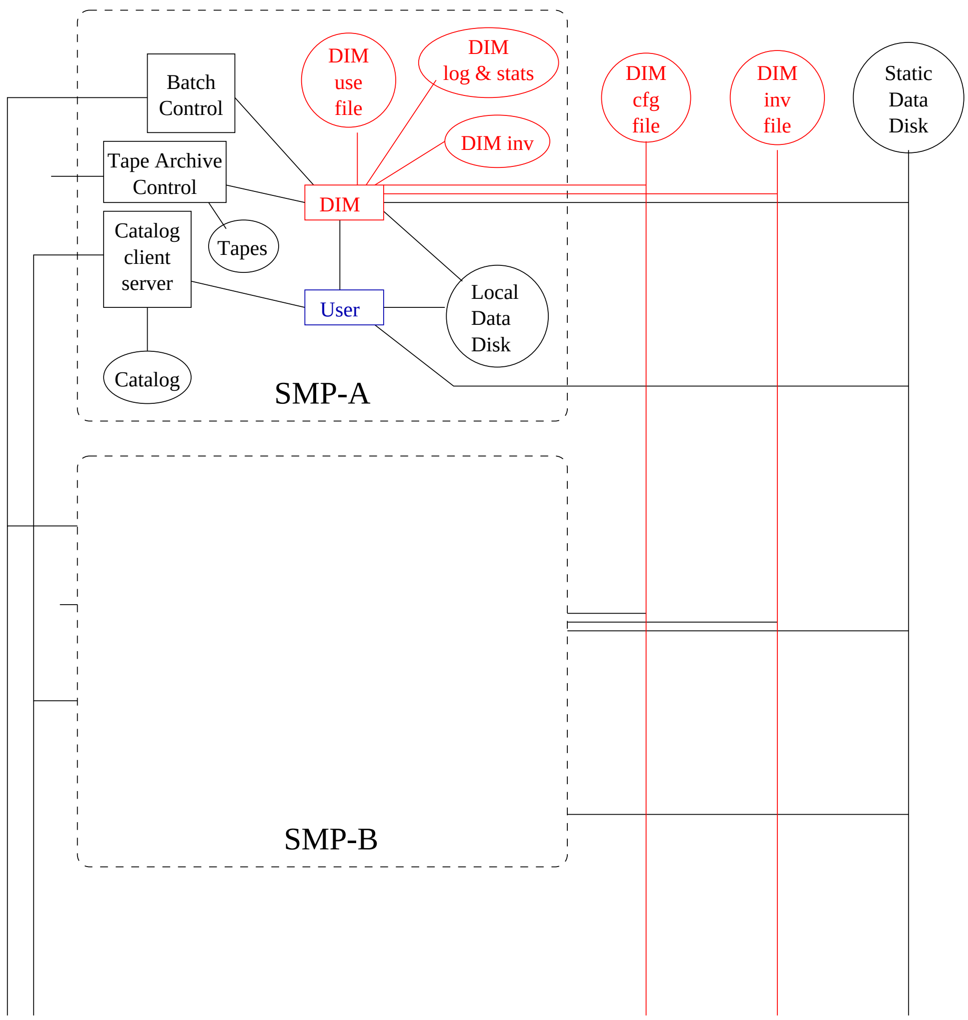

Figure 3: CDF Run II: Components of Resource Management Scheme in Baseline Hardware

```
- what data(sets) got what use?

- what disks got what use by what filesets/datasets?

- what data got fetched from tape how often?

- time plots of fraction of saturation in (disk, tape drives,
batch queue slots)

- history of stager actions (access to event logfile)

- which SMP has which datasets?

- job history files/dbase (what filesets were missed/processed?;
allow to be switched on/off by user; restart from crash.)

- improvements in LSF monitoring of batch jobs.
```

Other missing functions in DIM, or in other parts of the resource management scheme, were:

- If there are many filesets on disk in a dataset at the start of a user job reading that dataset, then all filesets get reserved currently even though they may not be needed by the user job for a long time. This action locks up disk space which might be a more valuable resource than tape drives.
- Daemons to clean up reservations or detect corruption of the DIM inv or use files will be needed.
- DIM actions need log files to trace troubles. So do the I/O stage input/output jobs (this was implemented in the prototype although printout from I/O jobs was minimal.).
- When an analysis job starts and finds no needed filesets on disk at that moment, it may be possible to re-queue that job until a fileset appears. This was not implemented nor tested in the prototype. It is not a part of DIM.

6 Other Elements of Prototype

6.1 User Job I/O Module in Prototype

As mentioned above, the user job in this prototype was implemented as a unix script. The general flow of actions in the user job is expected to be the same in the full CDF user job and is shown by the comments from the unix script:

```
#!/bin/csh
# ana_width.csh
#
# Translate dataset into list of filesets.
# Count filesets on disk.
```

```
#
# Reserve complete list of filesets.
# Launch io_queue jobs, on hold, to fetch missing filesets.
# Get list of io_queue job numbers.
#
# If too few filesets are on disk, then
#   bresume up to 2 io_queue jobs
#   and wait for at least one fileset to appear on disk
#
# = = = = = = = = =  TOP OF LOOP = = = =
# Loop to process one fileset.
# Process a fileset from list on disk
#  get path to fileset
#  get list of files in fileset
#  at last, process a fileset.
#  release this fileset
#
# Remove done fileset from unprocessed list.
#
# If (any filesets left to be processed) then
#    Get current list of filesets on disk from disk inventory.
#    bresume some io queue jobs, if necessary,
#    i.e. if disk contains:
#     >3 fileset, bresume no job
#      3 fileset, bresume  1 job
#      2 fileset, bresume  1 job
#      1 fileset, bresume  3 job
#      0 fileset, bresume  5 job
#    If current list of on disk is zero, wait.
#    goto top of loop to process one fileset
# endif
# = = = = = = = = = BOTTOM OF LOOP = =  = = = =
#
# If any io_queue jobs still exist, bkill them.
```

The flow of messages occurring in this flow of actions is shown in Figure 4.

As can be seen from the two figures, the algorithm for initiating fetches of filesets from tape to disk is implemented in the prototype by releasing batch jobs in the queue i_o_queue from suspension using the LSF command bresume; in the full DIM, this action would occur through DIM, not as direct communication with LSF.

6.2 CDF Datafile Catalog in Prototype

The CDF Datafile Catalog was implemented for the prototype using miniSQL. The essential use of the Catalog was to get the list of filesets forming a dataset. Here is a typical response:

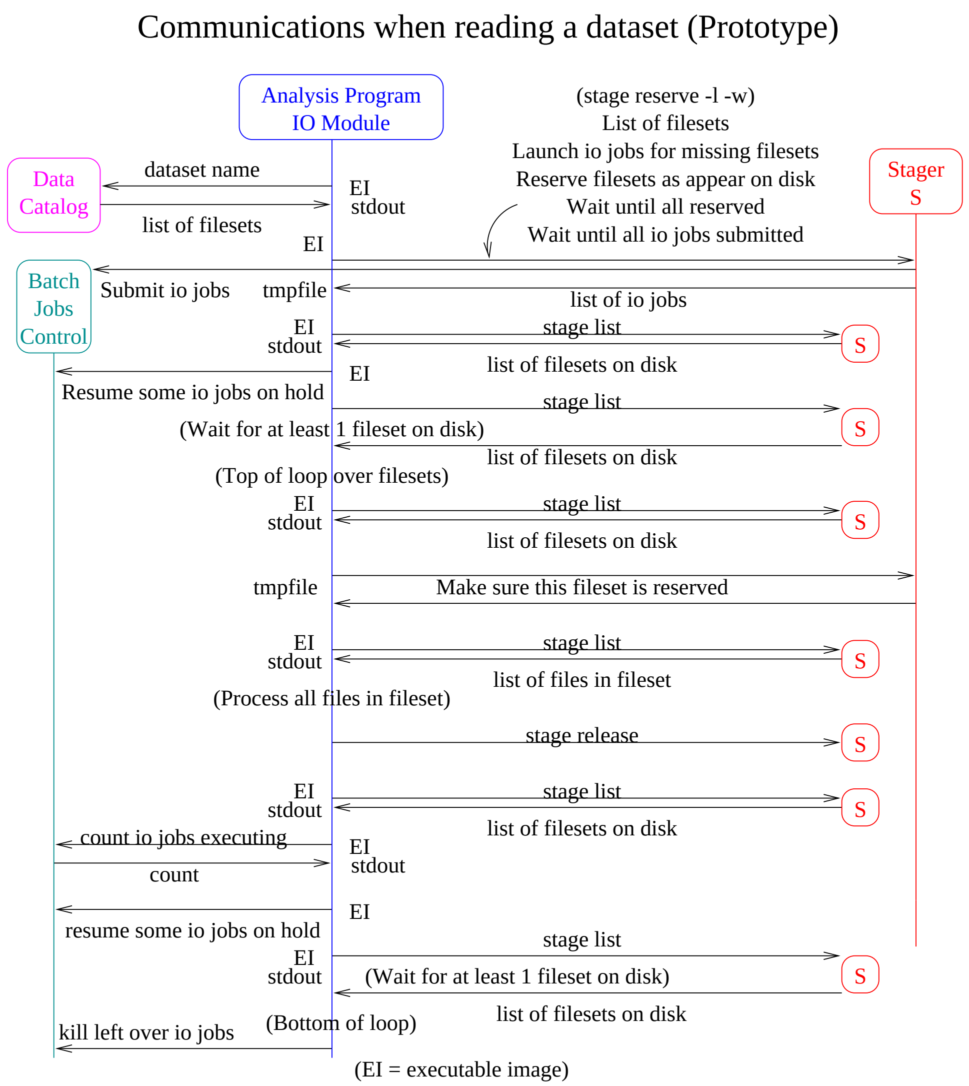

Figure 4: Flow of messages between components when reading a dataset.

```
% list_filesets.csh LONG_D
For dataset name LONG_D, fileset list is:
DMY040.0 DMY040.1 DMY041.0 DMY041.1 DMY042.0 DMY042.1 DMY043.0
DMY043.1 DMY044.0 DMY044.1 DMY045.0 DMY045.1 DMY046.0 DMY046.1
DMY047.0 DMY047.1 DMY048.0 DMY048.1 DMY049.0 DMY049.1
```

The fileset names differing only after the period symbolized filesets in different partitions on the same tape volume; the fileset name carried the tape volume name.

The complete set of datasets used in the prototype tests could be obtained by a database query:

```
Welcome to the miniSQL monitor.  Type \h for help.
mSQL > SELECT DISTINCT dataset_name FROM file
Query OK.  10 row(s) modified or retrieved.
 +--------------------------------+
 | dataset_name                   |
 +--------------------------------+
 | SHORT_A                        |
 | SHORT_B                        |
 | SHORT_C                        |
 | SHORT_D                        |
 | LONG_A                         |
 | LONG_B                         |
 | LONG_C                         |
 | LONG_D                         |
 | SHORT_E                        |
 | SHORT_F                        |
 +--------------------------------+
mSQL >
Bye!
```

7 LSF Batch System

The batch system used for the prototype is LSF (Load Sharing Facility) [2]. There are a number of reasons for choosing it for the prototype:

- It is already in wide use at FNAL (and at other HEP sites, e.g. SLAC, CERN) and was therefore available for us to install.
- It is a potential candidate for the Run II data handling system so we want to evaluate it.

LSF operates in a heterogeneous computing environment as is proposed for the Run II analysis system. Many different types of queues can be defined with different priorities and resources requirements attached to them, allowing us to make various types of dedicated CPU queues. It provides controls to prevent users from flooding the system with jobs. Various types of scheduling such as fairshare or the more simple first-come-first-served are provided. It is also possible to write a custom scheduler using the API that is provided should this be necessary. Availability of tape drives can be provided to the system as an external resources so they can be used in job scheduling if required. LSF comes with a good set of monitoring tools to allow the user to see the status of their jobs with both a command line interface and a GUI. It also provides accounting tools for administrators.

Evaluation of LSF is not finished at this time. Standard operations with LSF seem to work well; we might note the very useful "bhist -t" command and "xls-batch" GUI. The bhist command was also used to generate the figures from trials in Section 9. Crucial evaluations to see if the detailed management of priorities in both classes of queues (cpu and I/O) were not tried with the prototype yet. The important feature "fairshares" was given some study in Trial 18 below.

8 Simulation Scenarios

In the DH simulation, we exercised features and tested responses with a wide variety of user actions. There were particular scenarios (patterns of use by the collaboration) to evaluate, see below, and the list will probably continue to evolve. The following considerations led to the choice of scenario trials reported in Section 9.

A How do small dataset jobs compete against large dataset jobs, i.e. how little re-staging can be achieved for small datasets that are repeatedly used but not locked onto disk? (Locked datasets are labelled "static" in DIM.)

B The freight train mode (see http://runiicomputing.fnal.gov/sam/) has received much discussion in meetings among the Computing Division and the Collider Experiments. In this mode of operation of the DH system, several users process the same big dataset from tape, and the DH system is constructed to minimize the accesses to tape by forcing synchronised reading of data on all the associated user jobs.

This minimising of tape access appears built into the method of resource management proposed here. Thus, a job which starts on a big dataset fetches some filesets to disk from tape, then other jobs working on the same dataset but submitted later, work first on the same files on disk. In addition, the staging jobs submitted to the I/O queue to fetch further filesets from tape, use the Disk Inventory database and its reservation algorithm to avoid having more than one fetch of a fileset from tape to disk. Thus jobs using the same large dataset become synchronized just as in the freight train mode. Various trials below report just this sort of activity. Note that the various user jobs in general will not see the filesets in the same order, and sychronisation will not occur if jobs are submitted too far apart in time.

C During trials, a method of controlling the number of tape drives allotted to one cpu job ("width" of tape I/O) was added to the DH system. Various ways of doing this were discussed and one was chosen for testing in the prototype; it may not be the final method used. The method implemented used "just-in-time" fetches of filesets, i.e. the user job I/O module, after processing each fileset, inspected the disk cache for needed filesets and if too few, released stage input jobs from suspension in the batch queue; the precise algorithm

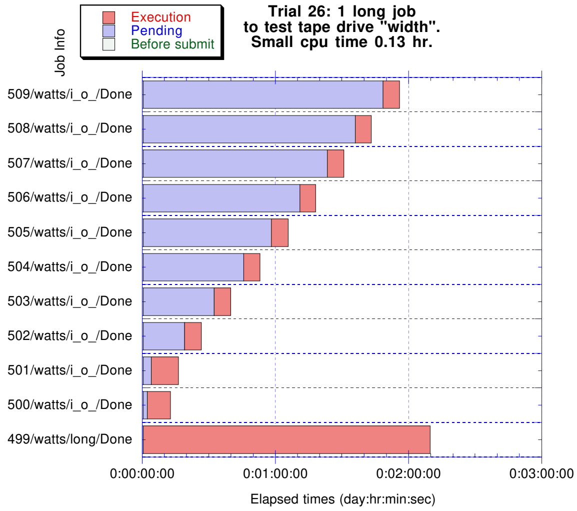

Figure 5: One long job showing just-in-time fetches.

can be seen in Section 6.1. It may be noted that this algorithm also conserves reserved space in the disk cache.

D What happens when users terminate jobs early or submit jobs which crash? What happens when resources are overloaded, e.g. when there is too much demand for disk space, or too much staging of the same data repeatedly.

Note that the protype long dataset jobs simulate actions by physics groups to regenerate secondary datasets from newly reconstructed primary datasets (although the output phase was not simulated). Short dataset jobs simulate users reading medium size secondary or tertiary datasets.

9 Results from Trials of Scenarios

9.1 Trial 26 - Demo of Just-in-time Fetches

One long job was submitted to show that the just-in-time fetches of filesets from tape to disk were occurring correctly. Figure 5 is a plot of batch job information against elapsed time. Divisions of the horizontal bars show time spent by a job pending in the queue and time executing. It can be seen that the I/O jobs do not saturate the two available execution slots. Detailed inspection shows that the fetches occurred at the correct times.

9.2 Trial29 - One long job vs a stream of short jobs

Figure 6 shows a scenario with a stream of jobs working on short datasets which were initially on the disk but were not locked there. The short jobs alternated on different datasets and were submitted by the same user, dhproto, at 15 min

Figure 6: Stream of short jobs interrupted by a long job and a robot job.

intervals; this 15 min interval is about 1/4 of the time to fill the staging disk from tape in the prototype so is equivalent to about 1/4 day for the full DH system. As can be seen from the figure, no staging jobs were necessary for this stream of short jobs, so the DIM kept track reasonably of use of datasets.

One job (544) working on a long dataset was submitted after about 2 hrs elapsed time and spawned a complete set of 10 stage input jobs. The long job competed for disk I/O and slowed down the short dataset jobs but forced no tape to disk transfers for them - a good result.

Another short dataset job (580) was submitted after 6 hrs which needed filesets not on disk. The resulting two stage input jobs also forced no extra stage input jobs for either the stream of short jobs nor the long job. The filesets for job 580 were on real tape in the Emass robot.

9.3 Trial30 - One long; stream of shorts; disk full

Figure 7 show a scenario where a long job (625) found most of its filesets already on disk and filled the disk with reservations (this feature was present in the prototype but probably will not remain in the full system). The following stream of short dataset jobs (628, 630, 632, etc) again alternated datasets and thus competed with each other for space on the disk. The I/O jobs resulting from the short jobs counted to 3 where only 2 were necessary and one per short job could have resulted for a total of about 20. Again the DIM reduced impressively the number of tape fetches.

9.4 Trial45 - Small freight scenario, 3 long jobs

Figure 8 shows a scenario of one physics group processing a primary dataset in 3 different jobs (3020, 3031, 3040). Even though the long dataset jobs are submitted 15 minutes apart, there are no extra fetches of filesets from tape to disk.

9.5 Trial18 - Big freight scenario, 10 long jobs

Figure 9 shows 10 jobs (2875, 2896, 2887, 2888, 2889, 2890, 2895, 2893, 2892, 2894) submitted over a substantial period of time at about one hr apart. (One other long job 2891 crashed with a bug; it does not affect these results.) All jobs read the same dataset LONG_A. A long dataset needs 10 I/O fetch jobs to get the whole dataset onto disk from tape; it takes about an hour to stage in a long dataset.

The 10 analysis jobs generated (10,3,2,3,3,3,4,3,3,3; sum=37) I/O staging jobs. Of the 37 I/O jobs, 25 fetched filesets and 12 did not (filesets were found to be already fetched by another I/O job). Thus there were 25 fetches.

The maximum number of fetches needed would occur if each job separately read tape, i.e. 10 jobs $\times$ 10 fetches = 100 fetches. This assumes no resource management scheme such as DIM.

With DIM, the minimum needed would occur if all 10 jobs were submitted at the same time and could execute simultaneously (10 executions slots); then only 10 fetches would be needed. With 4 execution slots and the fact that the disk has room for the results of 8 fetch jobs, the minimum needed is 10 for first 4 analysis jobs , 2 for next 4 analysis jobs, 2 for last 2 analysis jobs, total = 14 fetches.

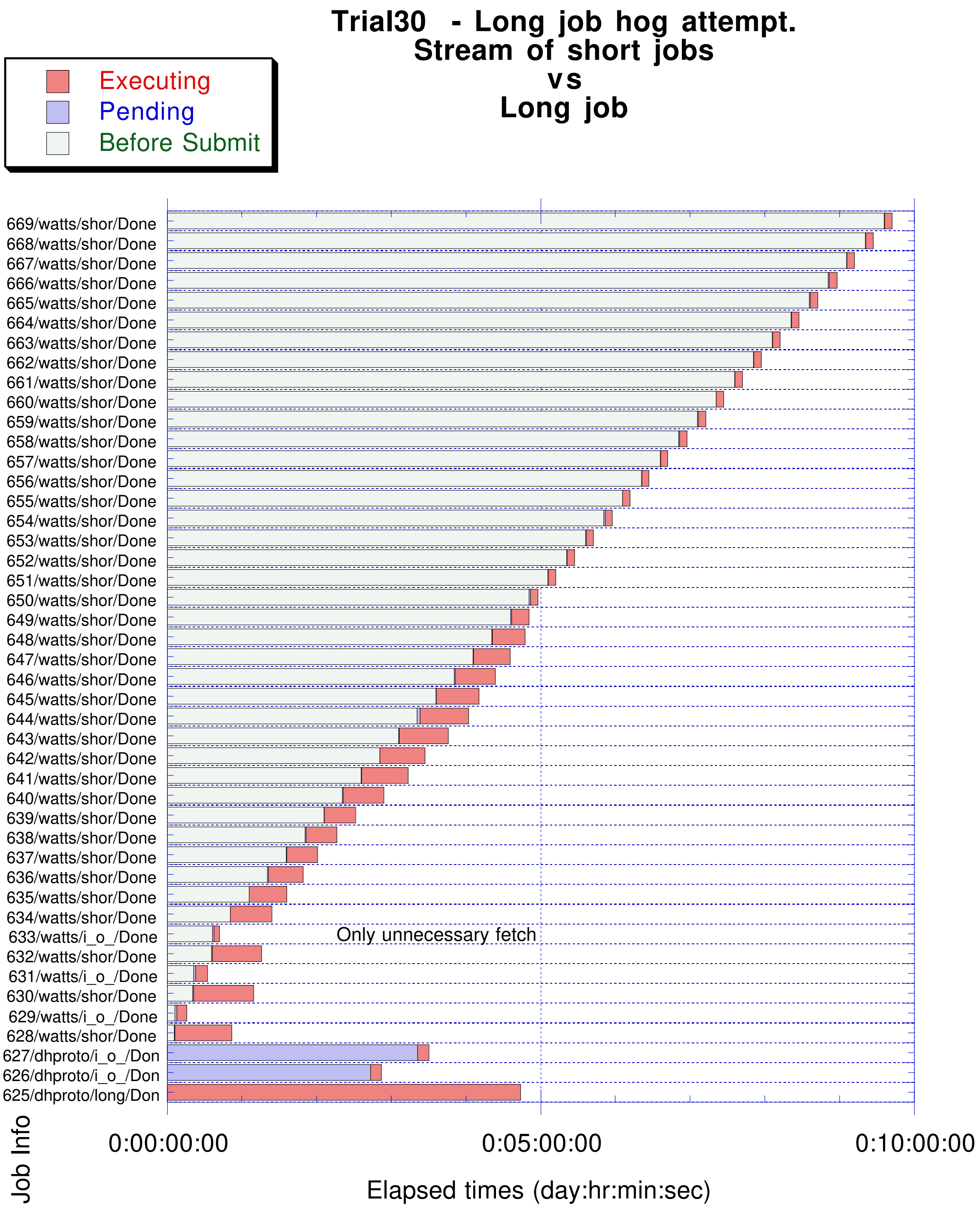

Figure 7: Stream of short jobs started after a long job filled disk.

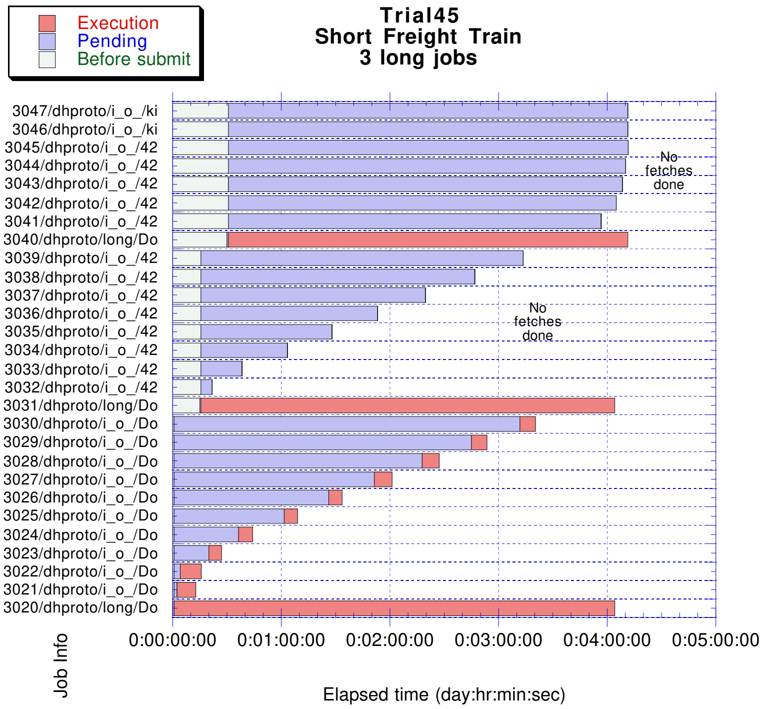

Figure 8: Small freight scenario, 3 long jobs.

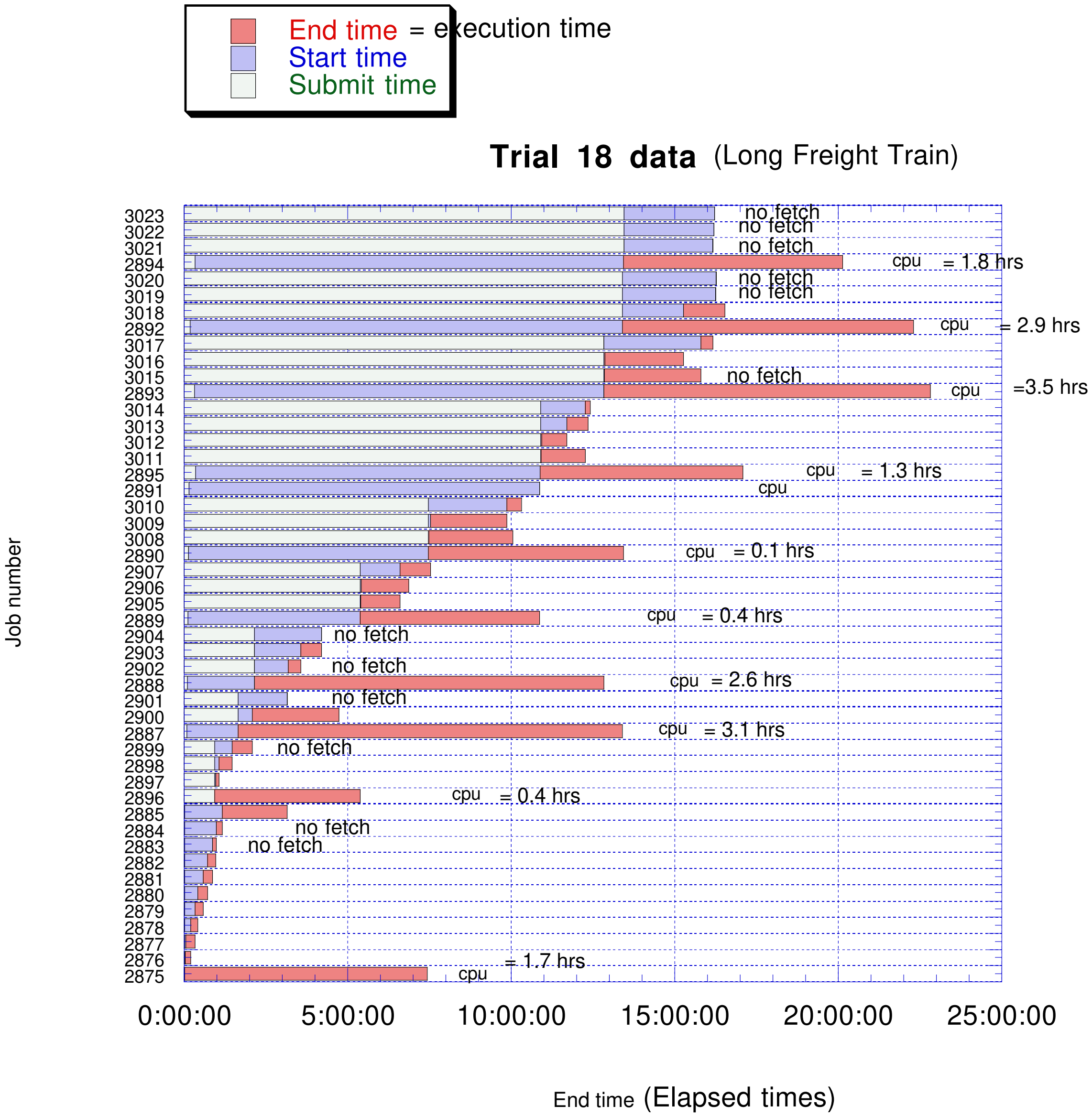

Figure 9: Big freight scenario of 10 long jobs.

If jobs run in isolation, and the disk remained unpurged, each would do 2 fetches per job, except job 1 does 10; total = 28. So the actual result of 25 fetches seems acceptable since the jobs were submitted with relatively long time between job starts.

Trial 18 was performed before just-in-time fetches were added to the algorithm. The lack of this feature can be seen dramatically in jobs 2876 through 2885; they are all executed too quickly for the long job 2875 to use their fetched data.

In this trial, the cpu use by each analysis job is noted in the plot.

9.6 Trial20 - Mixed set of long jobs and users

Figure 10 shows 6 long jobs by 6 different users reading 4 different long datasets, all submitted very close in time. There were (3,10,10,10,8,10; total 51) I/O stage input jobs for datasets (LONG_A,B,C,D,A,B). Thus there was some repetition of fetches for datasets LONG_A and LONG_B because the limit of 4 execution slots in the analysis batch queues prevented the 5th and 6th jobs overlapping with the 1st and 2nd. The prototype lacked a proposed feature of the resource management scheme to delay execution of jobs which find no data on disk which would have made 18 of 51 fetches unnecessary.

A feature of the LSF batch queue system was implemented in time for this trial, i.e. the application of "fairshares" among different users in the same queue. In the I/O queue for this trial, the scheduling of stage input jobs was more evenly distributed among different users than had been true in previous trials. This trial was also run before just-in-time fetches were implemented and the combination of both of these missing features in past trials had always led to execution of stage input jobs in order of ascending job numbers. The addition of fairshares in this trial clearly spread the execution of stage input jobs among users dhproto2 and dhproto3. However, user dhproto was given first priority and finished early. As explained in Section 7, more study of the usefulness of LSF is needed.

9.7 Trial40 - Stream of short jobs battles 4 long jobs

This scenario puts stress on the limited capability of the prototype scaled down computing system. All execution slots in the batch queues were filled almost all the time - 4 each in the short and long analysis queues, and 2 in the stage input i_o_queue, for a total of 10. The four long jobs read different datasets and were submitted by different users. These four jobs were submitted close together in time and finished at about the same time. The stream of short jobs by another user alternated between datasets SHORT_A and SHORT_D.

When the long jobs started, both short datasets were completely on disk. However, the demand for space by the long jobs forced 10 fetches of filesets for the short jobs; the extra fetch jobs are marked in the plots. There were about 24 short jobs submitted after the long jobs were submitted and before the long jobs finished. Potentially there were 48 fetch jobs needed (2 per short job) if the long jobs completely hogged the disk space, but 10 actually occurred. This seems acceptable. A somewhat bigger disk would have reduced this number of fetches to zero and obviously would be likely the policy adopted in the full DH system.

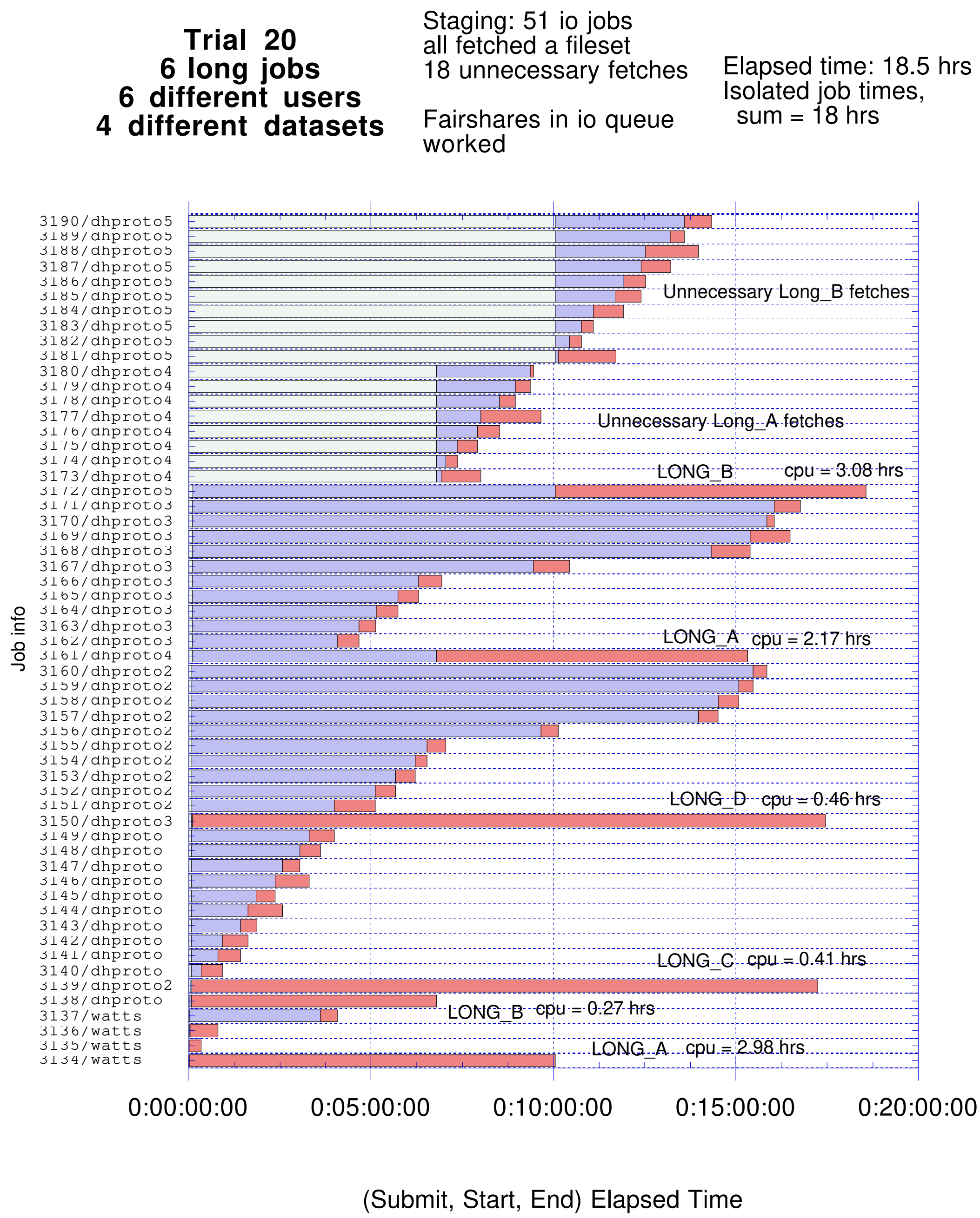

Figure 10: Mixed set of long jobs and users.

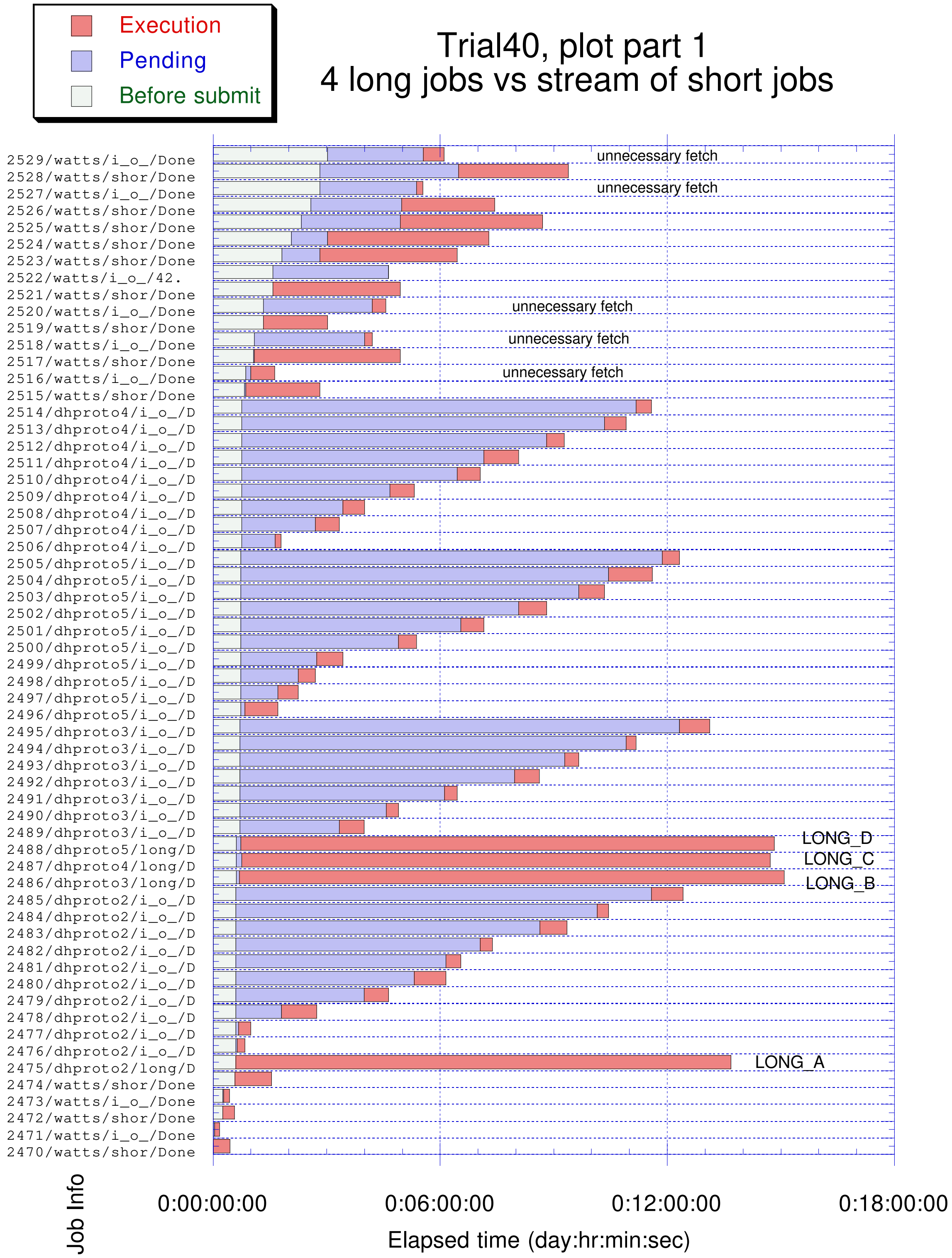

Figure 11: Four long jobs vs a stream of short jobs. First part of plot.

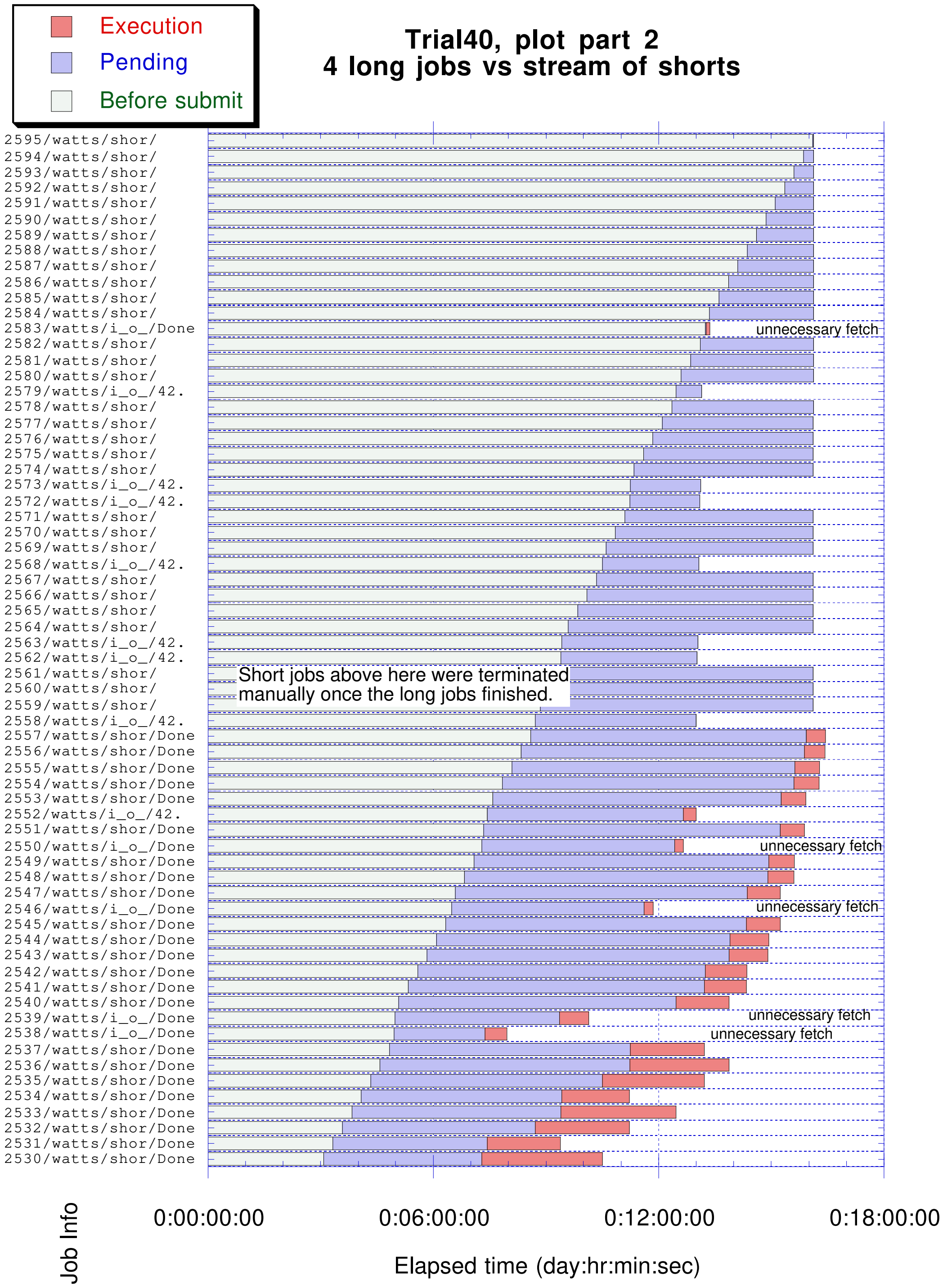

Figure 12: Four long jobs vs a stream of short jobs. Second part of plot.

This scenario gives much congestion in the operation of DIM since there are 10 jobs executing simultaneously on a single CPU machine, each with many copies of the DIM executable image. Several problems with functions such as stage input and stage list occurred only in this congestion. After fixing several, it was concluded that they were special to the particular implementation chosen for the prototype and further fixing would not teach us anything about the full client/server implementation. Thus, we stopped fixing these congestion-related problems. However, this trial 40 happened to run with no such problems.

10 Conclusions

The Disk Inventory Manager (DIM) worked well in prototype. The functions were appropriate, simple to understand, and worked well. The behavior shown in the mixes of types of user jobs ("scenarios") gave good understanding for future development of the full resource manangement scheme.

More evaluation of LSF is needed.

Acknowledgements

Marjorie Shapiro, Dan Amidei, and Steve Wolbers took part in many valuable discussions.

References

[1] W.Brown *et al.*, CDF Run II Data Handling, Resource Management and Software Components, CDF Note 4759(1998).

[2] The batch system used for the prototype is LSF (Load Sharing Facility) from Platform Computing Corporation (http://www.platform.com). Version 3.0 was used for early tests, version 3.2 for later.

[3] W.Brown *et al.*, CDF Run II Data Handling, Central Analysis System Hardware Architecture, CDF Note 4707(1998).

[4] E.Wicklund, Data-File Catalog for Data Analysis in CDF Run II, CDF Note 4761(1998).