TPM Production Manager Operator's Manual and User's Guide

Kirill G. Denisenko

D0 Note #2024

January 7, 1994

1.0 Introduction

This manual is intended to give a technical description of the TPM production manager running on the D0 SGI farm. The TPM is a collection of the c-shell scripts and executable code written in Fortran and C. Awk programming language is widely used. As opposed to the "How-To Operator's Guide To TPM" this manual requires basic knowledge of the Unix operating system, c-shell and awk language.

The structure of this manual is as follows: in Section 2.0 on page 3 we describe the data production model and hardware requirements. Section 2.2 on page 4 also covers the directory structure that is maintained on the local and shared disks. Section 2.7 on page 9 gives a description of the spooling system. Section 6.0 on page 18 covers the collection of scripts comprising the core of the TPM itself, with subsections on resource handling and job submission. The project handling is explained in Section 4.0 on page 15. Section 5.0 on page 16 describes various stages of bookkeeping. Section 3.7 on page 13 depicts the communications with the D0 file server (D0FS). Section 8.0 on page 18 is devoted to adapting TPM for streaming.

2.0 Production Model And Hardware Requirements

Since the size of the typical D0 event is fairly large (~500 Kilobytes) and the CPU time necessary to process an event is relatively small (~18 sec) the requirements to the I/O system are very high. There are several modes in which this production system may operate. The main, default, mode of operation is *parallel.* Two other modes, *stand-alone and sequential,* are operational and will be also described here.

The first model is operating in a so called *stand-alone* mode. It calls for each production CPU to be equipped with the local disk capable of holding the DST files for the particular job and two tape drives. The unit job hence becomes a tape. The two I/O processes, the inspooler and the outspooler, are running asynchronously with the production shell and communicate with it via the special disk files. As soon as the input dataset is spooled and the production shell finds it, the reconstruction code begins processing it. As soon as the reconstruction code finishes, the output STA file is spooled to the output tape. The DST files are being kept on disk until the whole job finishes and are transferred to the D0 file server (D0FS) afterwards. This mode is perfect for small special projects; it also doesn't introduce any overhead associated with serving events over NFS or TCP/IP.

The second, so called *sequential*, model moves from the stand-alone node idea to I/O server and worker nodes concept. The I/O server now is a computer that serves the devices used to spool the data in and out (disks and tapes). The worker nodes do all the processing and access the disks via the NFS on input and through the TCP/IP protocol on output - worker nodes send the processed events over the Ethernet directly to the client processes running on the I/O servers which in turn store events on the local disks. Amount of hardware necessary to operate in this mode though smaller than in a stand-alone mode is still quite large. Each worker node processes its own dataset which means that the amount of the spooling space is a multiple of the number of the worker nodes. In our case a system with 6 worker nodes requires at least 4 Gb of a disk space.

The third model introduced with RECO 10.13 is a truly *parallel* way of serving events over the network. On I/O server there are two processes for each logical machine: *insrv_d0.x* that read events from the input dataset and ship them to a worker node that processes an event and ship them via TCP/IP to another server process *outsrv_d0.x.* See Section 2.3 on page 6 for more information.

2.1 TPM Basics.

Here I define the notation for the environment variables that will be used in the following text:

$HOME environment variable is pointing to the home directory of *"dzero"* account which can have different names when mounted on different computers.

$PRODIR points to *$HOME/proman* directory.

$FMLNODE is set to the name of the logical machine in question, for example *fnsfe_1* in case where there are more than one logical machine per physical CPU or *fnstfj* where logical machine is equivalent to the physical CPU.

$PRODAT environment variable is defined in a *resource.$FMLNODE* file and points to the spooling disk directory from where the rest of tpm directories is hanging.

TPM production manager serves is a system that maintains the production resources (input and output tapes, tape drives, disks and CPUs), submits jobs for processing and does the bookkeeping. The unit job for TPM is a tape (each job is assigned a unique number, called *queid*). The unit processor is a logical machine, i.e a group of worker nodes working on the same project (see Section 4.0 on page 15 for the definition of a project). Each logical machine is assigned at least one output spooler (see Section 2.7 on page 9 for details how spooling is done). A logical machine is declared via *resource* and *resource.LM* parameter files (see Section 2.3.4 on page 6 for details about these parameter files). Input tapes are obtained from D0FS databases using *$PRODIR/exe/get_new_jobs* script. They are entered into *$PRODIR/resources/injobs* (see the following Section 2.2 on page 4 for the definition of the environment variables used; environment variables are denoted by a leading $). Each input tape is assigned a project from a pool of available projects according to the first three letters of the filenames on tapes by *$PRODIR/exe/get_new_jobs*. In the *$PRODIR/resources/resource* a logical machine is assigned a project and an input spooler number. When the current job processing on a particular logical machine finishes the TPM production manager will look for a tape with a project matching one assigned to this logical machine in $PRODIR/resources/injobs. The tapes waiting to be processed are denoted by **WAITING** status in this file. If such tape exists it will be submitted for execution. A script *$PRODIR/exe/jobcontrol.csh* is running every day on the NFS-disk server (currently *fnsfg*) and picks up datasets missing from the already processed tapes. Tapes that re-submitted are denoted by the word **REPROCESSING** in the injobs file. Per each resubmitting one **REPROCESSING** would appear.

2.2 Disks Directory Structure.

2.2.1 Database-like directories.

First I introduce the underlying structure of the many directories that contain the log files for the TPM production manager. A good example of these would be the *$PRODIR/logdb* directory which contains the log file for the I/O servers and reconstruction shells (the contents of it will be discussed elsewhere in this document). Since Unix operating system cannot efficiently handle the directories with more than 200 - 300 files in them TPM stores these files according to the time when they were produced.

There is a subdirectory corresponding to a year (4 digits), month (three starting lowercase letters), and a 6 day period within a month (1-6, 7-12, 13-18, 19-24, 25-31). Thus for a file produced on May 14, 1993 the log files can be found in a subdirectory *$PRODIR/logdb/1993/may/13-18*. The rest of the database-like directories have the same underlying subdirectories.

2.2.2 Disk structure of the local spooling disk.

The local to the production node disk should have the following directory structure:

- **$PRODAT/$FMLNODE/inspool** is the area where the inspooler process will spool the raw data in the *parallel* mode.
- **$PRODAT/$FMLNODE/raw** is the area where the raw data is being placed for processing.
- **$PRODAT/$FMLNODE/proman/dst** is the area where the DST files are kept before transferring to the D0FS.
- **$PRODAT/$FMLNODE/proman/sta** is the area where the STA output goes before spooling to the output tape.
- **$PRODAT/$FMLNODE/proman/ linktemps** is the area where the temporary links to the DSTs are stored when zftp'ing them to the D0FS.

All these directories are being cleaned by the production manager before starting a new job.

The server node (currently fnsfg) also has disks shared with the rest of the production nodes. These are:

2.2.3 TPM Code and Bookkeeping Disk

"$HOME" disk contains all the production manager software, all bookkeeping software and files, and all the log files. This disk is NFS-mounted on the rest of the production nodes.

- **$HOME** home directory contains .netrc file which has a login information necessary to transfer files to the D0FS (D0FS account DZERO is used, password.........). It also contains .rhosts file that should have all the authentication info for the worker nodes.
- **$HOME/proman/exe** is the directory where the executables and scripts can be found (in alphabetical order):

  after_crash Restarts the TPM system after the reboot. Also restarts the dbl3 servers and picks up all failed network transfers and bookkeeping processes.

  analyze_list.nawk Part of the package of scripts for updating an input queue.

  analyze_project.nawk Same as above.

  append.x Allows you to append to a file that may be shared between different processes without a danger of overwriting.

  check_buffer.csh Checks that the D0$DATA$BUFER has enough disk space to accept the incoming DST files.

  check_disk_space Used in *sequential* mode to avoid the disk space overflow.

  check_invol.csh Checks that the input tape is not in use on D0FS. See Section 3.7 on page 13.

  cli_d0.x Client executable which is used in sequential mode to receive events from the worker nodes and store them on disk.

  copy_results.csh A top level script to copy the DSTs to D0FS and create and copy RCP-files to D0FS.

  full_d0reco_V_P.x D0RECO reconstruction program. Here V stands for its version, and P - for the pass release.

  inspool_l.x Input tape spooler executable.

  insrv_d0.x Input parallel server executable

  jobcontrol.csh Recovers the datasets that were lost the first time and resubmits them for reprocessing.

  outspool_mu.x Output tape spooler executable.

  outsrv_d0.x Output parallel server executable.

  pick_up_all Checks for any failed network transfers and bookkeeping processes on all TPM-related I/O servers and restarts them.

  pick_up_trans Checks for any failed network transfers and bookkeeping processes on a CPU-node it is invoked and restarts them. Called from *pick_up_all.*

  prl_check.csh A script that periodically checks the status of the worker node *wrkshell*'s and resubmits them if necessary.

  shift_alarm.csh A script to page the shift pager. The argument is a code that will be displayed on the pager.

uptest Executable that tests the network connection between D0 Unix farm nodes and D0FS. It also performs writability tests of the involved areas on D0FS (D0$DATA$BUFFER, FMD0:[RCPFARMIN], and PROD_DB:[SGIDATA]).

zftp Custom copy of the CERN utility.

2.2.4 DBL3 Database disk (/dbl3)

"/dbl3" disk contains the DBL3 databases and the DBL3 server code. This disk is NFS-mounted on all of the production nodes and on the local D0 SGI computers. The top level directory contains dbl3 server executable, *d0dbl3_server.exe*, and scripts to start adbl3 servers, *start_all_servers,* and shut them down, *kill_all_servers*. *who_is_running* script (aliased to *showdb*) is used to display a current status of dbl3 servers. The disk is further partitioned per detector.

/dbl3/cal

/dbl3/dbm

/dbl3/cdc

/dbl3/fdc

/dbl3/muo

/dbl3/sam

/dbl3/trd

/dbl3/vtx

The database for each detector can be found here. File *DET_DBFILES.DAT* defines the run validity ranges. In addition to this these subdirectories are further partitioned into two subdirectories: *server* and *todo*. *server* contains the dbl3 server setup files and server log file, while *todo* is used to receive the database update files from D0FS.

2.2.5 Summaries And Histograms Disk (/proman)

Disk "*/proman*" is currently used for the storage of the D0 reconstruction code (D0RECO) summary files (*/proman/sumdb*) and files containing histograms (*/proman/hstdb*) produced by the D0RECO program. Currently they are merged per run. Both of them has the database underlying structure as described at the beginning of this Section 2.2 on page 4. See Section 5.2 on page 17 for details.

2.3 How a Parallel Shell Is Organized

This section describes a *parallel* shell (which is now the default mode of operations on a D0 Unix Farm)

Files involved are described in subsequent subsections.

2.3.1 d0reco_prl.csh

in *$PRODIR/proman/tpm*. It is the shell that is running one per logical machine on I/O servers (currently *fnsfe_0*, *fnsfe_1, fnsfe_2*, and *fnstfj*). It is submitted the regular way by the TPM production manager. It handles the submission and control of the I/O server executables *insrv_d0.x* and *outsrv_d0.x* and moving around raw and processed files.

2.3.2 wrkshell

in *$PRODIR/proman/tpm*. It is a shell script that is running all the time on all worker nodes and submits *d0reco.x* executable when it sees that an appropriate in- and output servers are running on the I/O server. There is a script that runs 15 minutes past every hour, *$PRODIR/exe/prl_check.csh*, that checks the status of the wrkshell on each of the worker nodes belonging to a I/O server declared parallel.

If due to whatever reason you will need to stop all wrkshell's for a particular logical machine you can place a file *stop_it* (contents of which are irrelevant) into *$PRODAT/$FMLNODE/project* area. Please note that the wrkshell will not exit until the *insrv_d0.x* and *outsrv_d0.x* servers are running for the logical machine in question.

2.3.3 Executables insrv_d0.x and outsrv_d0.x

reside in *$PRODIR/exe*. The source code of them can be found in *$PRODIR/source*. You cannot link on *fnsfe* so it is provided for reference only.

2.3.4 Parameter files.

To declare server to be parallel you set MODE to PARALLEL in a file *$PRODIR/resources/resource.$server*, e.g. *resource.fnsfe_3* reads

PRODAT_DIRECTORY /spool02/dzero

MODE PARALLEL

fnsf128 READY

fnsf129 READY

fnsf130 READY

fnsf131 READY

fnsf132 READY

fnsf133 READY

fnsf134 READY

fnsf135 READY

If you want to take out a particular worker node change its status from READY to UNAVAILABLE.

All the standard parameter files (resource, etc.) apply to a parallel mode.

2.3.5 d0reco_rcp for a parallel shell

In order for the D0RECO executable to recognize its mode of operation as parallel it is necessary to include the following line into the *d0reco.rcp* file:

PARALLEL .TRUE. ! use network to serve events

The location of the *d0reco.rcp* file is defined in the project description file. See Section 4.0 on page 15.

2.3.6 remote_rcp for a parallel shell

A file *remote.rcp* is used by both D0RECO executable and parallel servers executables. It contains the TCP/IP port definitions. Its proper place is in the *project* area on the spooling disk, *$PRODAT/$FMLNODE/project*.

2.4 Diagnostics of a parallel shell behavior.

2.4.1 Symptoms.

Please not less often than 3-4 times a shift check the area *$PRODAT/$server/proman/sta*. This area normally looks like following:

fnsfe/spool02/dzero/fnsfe_3% ls -l proman/sta

total 586547

-rw-r--r-- 1 dzero d0 120 Apr 20 14:00 4795_014_063797_52.ous

-rw-r--r-- 1 dzero d0 165012120 Apr 20 14:00 4795_014_063797_52.sta

-rw-r--r-- 1 dzero d0 135298800 Apr 20 14:13 4795_015_063797_53.sta

Here please notice the timestamp on a file that doesn't have a corresponding **.ous* file. Repeat it in 10-30 sec and then make sure that this file grows in size and the time stamp is updated. This is enough of an indication that everything is fine with a particular logical machine.

If there is nothing in this area check that the input tape spooler is running and there are files in *$PRODAT/$server/inspool* and $*PRODAT/$server/raw*. If there are files paired with **.done* files in either one of these directories then make sure that *d0reco_prl_$server* is running. If it is, you need to restart the server.

I want to notice that normally you won't have to do it; a mail message *"Parallel servers on $FMLNODE will be restarted"* is an indication that the server took care about itself.

If you see some files in *$PRODAT/$server/proman/sta* area but they are stale (the file without **.ous* is not growing or there is no such file) proceed checking as described above.

2.5 How to restart a parallel server.

1. Make the I/O server UNAVAILABLE in a resource file. You have to be on fnsfe and make sure that sleep 800 is showing

2. rlogin into the machine in question.

3. Kill the *d0reco_prl_$server* prl shell if it is running.

That is all you need do to. Logout back to fnsfe and declare a logical machine FARMLET back from UNAVAILABLE. In a little while check the *$PRODAT/$server/proman/sta* area again.

!! I want to underline that it is very unlikely that you'll have to do it. Please make sure that it is warranted!!

2.6 What is the meaning of all these *.ios and *.ous files

2.6.1 *.ios

When a parallel shell sees a file in an *$PRODAT/$server/inspool* area it moves it into the

$PRODAT/$server/raw area and starts input and output server for it. When the input server gets a connection from the worker nodes it starts to look for a file already prepared for it by a parallel shell. The file that contains the name of the file to be processed is pointed at by an environmental variable **INLIST**. An input server then creates a file with extension **.ios* that contains the name of the file being process by a server. There may be several such files (but no more than 3) at any particular moment, especially when the run number is switched and a new group of servers is started. The format of the file is *QUEID_SFNO_-RUN_PART.ios*, e.g. *4821_002_063824_22.ios*. Here QUEID is the regular tpm queue id no (unique for each input tape), SFNO is a file number for the current server (the sequential number if these file in the stream of files processed by this particular copy of the server), RUN and PART are self-explanatory. The dst and sta files are initially indexed the same way with extensions.dst and.sta correspondingly. They are renamed according to a standard Dzero convention when the **.ous* file appears in **sta** directory.

2.6.2 *.ous

A file with **.ous* extension file indicates that processing for a particular dataset is finished and the file is closed. **.ous* file contains information about the starting and ending event number and number of lost events (if any). This file is used to create RCP file and deleted afterwards together with raw data file and **.ios* file.

Format of **.ous* file is as follows:

N_TOT_EVENT 231
N_MIN_EVENT 4853
N_MAX_EVENT 5083
N_MISS_EVENT 0
L_MISS_EVENT

2.7 Overview of spooling as it is done on D0 Unix Farms

The spooling (copying input files from tape to disk and copying output files from disk to tape) is completely asynchronous with data processing. New files are copied into the designated inspool areas when there no previous files with *.done extension there. Output files are copied when **.done* files appears in the outspool area. Both input and output spoolers are started by *“start_farmlet_job”* script which in turn is called by *“update_farmlet_job”* script.

2.7.1 Input spooling.

An input spooling is a process of reading the input tape and writing the resulting files into the specified area. The spoolers use FNAL supported RBIO package (v6_2e currently) to do basic I/O operations on tape and disk. Tape mount requests are supported by FNAL CPS package.

2.7.2 Assignment of the input spoolers.

You can assign any number of spoolers to serve any number of spooling areas. Thus you can request one input spooler to serve several logical machines or, alternatively, several input spoolers to serve one logical machine. The most common setup though is when one input spooler is serving one logical machine.

The actual assignment happens in the *$PRODIR/resources/resource* file where you specify the input spooler number in the fourth column. This number can be the same for any number of logical machines (running on the same physical machine, of course) in which case they will all be served by the same server. An example follows:

NODE	PROJECT	STATUS	SPOOLER #
fnstfj	D0GEANT_SGI	FARMLET	2
fnstfk	RECO_FULL_SGI	FARMLET	2
fnsfe_0	RECO_FULL_SGI	FARMLET	4
fnsfe_1	RECO_FULL_SGI	FARMLET	5
fnsfe_3	RECO_FULL_SGI	FARMLET	3

Please note than when the same number is given for logical machines running on different physical I/O servers it doesn’t mean the same input spooler, of course. In this example, all logical machines are served by their own I/O servers.

The input spooler number actually points the system to a file in *$PRODIR/resource* directory. This file has a name in a following format:

inspool_list_SPNO.SERVER

where SPNO is an input spooler number as specified in the resource file, and SERVER is a physical server name. For instance, in the above example of a resource file an input spooler number 5 is specified for as logical machine *fnsfe_1*. That directs tpm production manager to look for a file

inspool_list_5.fnsfe

The contents of this file represents the list of directories (that should include absolute path) on a particular machine to spool in:

/spool01/dzero/fnsfe_1/inspool

This line tells the TPM to spool into this area. You can have as many areas as you want per each file which is necessary when you operate in a SEQUENTIAL mode:

inspool_list_1.fnsfe has, for example, the following contents:

```
/spool00/dzero/fnsfe_0/fnsf112/proman/inspool
/spool00/dzero/fnsfe_0/fnsf113/proman/inspool
/spool00/dzero/fnsfe_0/fnsf114/proman/inspool
/spool00/dzero/fnsfe_0/fnsf115/proman/inspool
/spool00/dzero/fnsfe_0/fnsf116/proman/inspool
/spool00/dzero/fnsfe_0/fnsf117/proman/inspool
/spool01/dzero/fnsfe_1/fnsf118/proman/inspool
/spool01/dzero/fnsfe_1/fnsf119/proman/inspool
/spool01/dzero/fnsfe_1/fnsf120/proman/inspool
/spool01/dzero/fnsfe_1/fnsf121/proman/inspool
```

This file covers two logical machines *fnsfe_0* and *fnsfe_1* for SEQUENTIAL mode processing.

2.7.3 How the system knows when to restart an input spooler.

When the TPM production manager is looping around the logical machines on a physical machine it gets the number of the input spooler assigned to a particular logical machine and restarts it if the previous input tape is finished or has failed. If one input spooler is assigned to several logical machine it will be restarted when the first machine on the list is processed.

2.7.4 Files with input tapes directory.

Please note that the input spooler is always looking for a file *tapes_moved.VSN*, where VSN is an input tape label. A normal location for this file is in *$PRODIR/history* subdirectory (see Section 2.2 on page 4 for the definition of *$PRODIR* environment variable). It will not spool the tape unless this file exists. You don't have to worry about it when doing regular processing because this file is created automatically by create_tape.csh script (in *$PRODIR/tpm)*. In special cases (like Monte-Carlo data or when you want to skip some files) you may want to create this file yourself. You can either create an empty file *tapes_moved.VSN* (in which case all the files on this tape will be spooled) or type in the actual contents or partial contents that you need to spool in a following format:

VSN SEQ FILENAME

Here VSN is a tape label, SEQ is a file sequence number on tape, and FILENAME is a tapefile name. For example, for tape WM3898 file tapes_moved.WM3898 would look like

```
WM3898  1 ALL_063797_39.X_RAW01
WM3898  2 ALL_063797_40.X_RAW01
WM3898  3 ALL_063797_41.X_RAW01
WM3898  4 ALL_063797_42.X_RAW01
WM3898  5 ALL_063797_43.X_RAW01
WM3898  6 ALL_063797_44.X_RAW01
WM3898  7 ALL_063797_45.X_RAW01
WM3898  8 ALL_063797_46.X_RAW01
WM3898  9 ALL_063797_47.X_RAW01
WM3898 10 ALL_063797_48.X_RAW01
```

This file *tapes_moved.VSN* is also used to remove a bad tape (a tape for which the input spooler fails over and over again) from the input queue. You simply type

rm $PRODIR/history/*.VSN

2.7.5 Output spooling.

Output spooling is a process of reading STA files from designated disk area(s) and writing them out to an output tape. As well as an input spooler output spooler does it basic I/O using RBIO package.

2.7.6 Output spooler assignment.

Each logical machine is always assigned an output spooler. There is no inflexibility in this since by definition a logical machine is a collection of worker nodes with the throughput that requires one output spooler to keep up with the data flow. The outspooling area thus is always defined as *$PRODAT/$server/proman/sta*; it is specified as an argument to outspool executable.

2.7.7 Output tape directory.

At any given moment of time you can always find an output tape contents by typing the *$PRODIR/pdbkd/$YEAR/$WEEK/STAT_VSN.RCP* file. Here $YEAR is a current year (e.g., 1993), $WEEK is a 6 day range (1-6, 7-12, 13-18, 19-24, 25-31) when the tape was processed and VSN is an output tape label.

2.7.8 Tape Drives Handling.

For each physical I/O server that is a part of the TPM production manager there is a file *drives.cpu_node* residing in *$PRODIR/resources* subdirectory. This file lists all the available tape drives and their status. To remove a tape drive from the spooling system change its status to READY. To add a tape drive to the spooling system change its status to UNAVAILABLE. Please remember that all these tape drives, both READY and UNAVAILABLE, should be included into the cps tape database for a production system in question. Below is an example of *drives.fnsfe* file:

```
sgi80  READY 0 2
sgi81  READY 0 0
sgi82  READY 0 1
sgi83  READY 0 1
sgi84  READY 0 3
sgi85  UNAVAILABLE 5 5
sgi86  READY 0 0
```

The third parameter denotes the number of fatal tape errors for this tape drive; if it is greater than 4 a tape drive is automatically set to UNAVAILABLE. This counter is reset after each successful session. The fourth parameter is a cumulative error counter, i.e. it is not reset after each successfull session.

2.7.9 Executables.

The current executables are in $PRODIR/exe. Their filenames are *inspool_l.x* for an input spooler and *outspool_mu.x*. The source code of them for reference purposes can be found in *$PRODIR/source.*

3.0 TPM Operator's Manual

The following is supposed to give a brief overview of the TPM - a production manager system running on the D0 SGI Reconstruction Farm from its operator point of view. The purpose of this system is to submit, monitor, and report the status of the reconstruction jobs, also to communicate with the outside world by spooling the DSTs to the DST area on the D0FS and preparing the RCP-files to make the corresponding entries in the FATMEN catalogue. TPM allows for handling of the several reconstruction projects (the number is limited only by the number of available cpu-nodes). The operator actions include monitoring the system status, handling of error conditions that require human intervention, and fulfilling requests for processing of the specified data. TPM will automatically recover from the majority of error conditions including, but not limited to, spooler hang-ups, reconstruction code failures, bad input and/or output tapes, etc.

This is not intended to be a complete technical manual on TPM. This document is a how-to manual for the operator and doesn't require any knowledge of Unix, c-shell, or awk-language. Please read the rest of these document to get more insight on TPM intrinsics.

3.1 How to log into the D0 SGI farm.

From any available computer type

telnet fnsfe

Answer dzero at *login:* prompt. The password is.........

3.2 How to check what the system is doing.

Type **tpm_disp** at fnsfe% prompt. This command will also work on all of the rest of the nodes of the D0 SGI farm (currently fnstfj and fnsfg). The resulting display will show the status of the last twenty one jobs. If you want to get the status for more jobs (going farther into the past) supply the command with an argument, e.g. **tpm_disp 40**. This will display a status for the last 40 jobs. The format of the table includes the job queue ID number - a unique number for any job, the farm node on which the job was (is) executed, the reconstruction project name, the input and the output tape volume label, the date this job was submitted, and the current status of the job. The status field for the running job may be:

SUBMITTED - a job has been submitted by the TPM. No communications with the job have been made.

RUNNING - the recently submitted job has started successfully on a given node. Both in- and out-spoolers are running.

NEXT_DS_OK - next input dataset was processed successfully, the job is still running since there is another dataset on the input tape.

RUN_DS_FAIL - next input dataset failed to be processed successfully. No harm to the job itself was done and no operator action is required. It is prudent though to look into the causes of the failure (see the diagnostics section).

INQUEUED - indicates that a job was submitted to a farmlet - I/O server with more than 1 worker CPU node. Currently 1 stand-alone node (*fnstfj*) and 1 farmlet (*fnsfe* with 24 worker nodes) are available. The status field for the finished job may be:

SUCCESS - all the input datasets for the job have been processed successfully. This is the most common situation.

FINISHED - at least one of the input datasets failed to be processed successfully. Look at the log files to find which and why (see the diagnostics section).

ENDED - an input tape was successfully in-spooled but no conclusion can be made about the status of the dataset processing. This is a standard message in the *parallel* node.

FAILED_ON_SP - an input or output spooler failure. Happens when there are problems with input and/or output tapes, or the tape drive hardware. Look into the log files for details (see the diagnostics section).

INTAPE_IN_USE - an input tape you want to use for the reconstruction is already used on D0FS. One important caveat - make sure that it is not you who are "using" this tape. If the node fails due to the local power outage or the hardware malfunction it cannot communicate to the D0FS that the tape in question has been released. You have to check it yourself - see "Communicating with D0FS" Section 3.7 on page 13.

There may be non-standard status messages (put in by hand by the farm manager), which are usually self-explanatory, e.g. **POWER_OUTAGE** would mean that the job in question failed due to the power outage.

Another command, **tpm_wide**, in addition to the goodies that **tpm_disp** provides also gives you the finish time of the job. **tpm_wide** requires 132 column output device.

3.3 How to add new blank tapes to the system.

All the following commands are valid in the *~/proman/resources* directory. After logging into fnsfe type

```
cd ~/proman/resources
```

To add tapes to the pool of available tapes do the following[1]:

```
add_blanks SERIES FIRST END
```

Here SERIES is a two letter series code (e.g., WN), FIRST is a starting tape number, and LAST is an inclusive ending tape number. So the real commands will look like

```
add_blanks WN 123 234
```

and will add tapes WN0123 through WN0234 to the pool of available tapes.

CAUTION! This script will not check for duplicate entries; it is your responsibility to make sure that the tape in question has not been already entered into the database.

1. You can type this command any time; you don't have to synchronize it with the production manager.

To check the last n tapes in the "blanks" database you can issue the following command:

tail -n blanks

3.4 How to enter new jobs to the pool of the jobs waiting for the execution.

Normally you don't need to do this since the entering jobs into the input queue is automated vi the get_new_jobs scripts which is running automatically every hour and checks D0FS for new input tapes vaulted. But in case a necessity would arise the following procedure would allow to add a tape manually. All the following commands are valid in the *~/proman/resources* directory. After logging into fnsfe type

cd ~/proman/resources

The unit job in our reconstruction model is an input tape. Hence "to submit a job" means to let the system know that a particular input tape is available for the reconstruction. In order to do this issue the following command:

injob_create

You will be prompted for the project name - the default one is RECO_FULL_SGI, just type <CR> to get it. Then you will be prompted for the input tapes labels. Enter <CR> after each tape to be added; enter <CR><CR> after the last one you want to add. After that the contents of your update will be displayed for your review. If you like it - type <CR> or Y at the prompt. If you don't - type anything except <CR> or Y. The program will then exit without updating the database.

- CAUTION! No attempt is made to check for the duplicate entries. The need to enter any of the input tapes twice may arise only to reprocess some of the datasets - an operator is not supposed to do this. In order to check which n jobs were entered last issue the following command:

  tail -n injobs

3.5 Reports

The reports generators are in *$PRODIR/report* area. To get there type the following command after logging to the farm:

cd ~/proman/report

The following reports are currently available:

1. Official farm report displays the output tapes available, run numbers for the datasets processed, date this job was submitted, number of events on a tape and total number of events processed. In order to obtain it type

rlogin fnsfg

official_report Mon STR &

exit

Here Mon is a three letter abbreviation for the month you want to obtain the report for (e.g., Jan, Feb, Oct, etc.)[1]; STR is a stream name (e.g., ALL, EXP, etc.).[2] The stream argument can be omitted in which case processed datasets for all available streams will be reported. The result can be found in the file

$PRODIR/report/log/off_rep_STR.MM_DD_YY

where STR is a stream name and MM_DD_YY is the today's date. It takes from 1 to 8 hours to process this report, depending on the node load.

Please also note that this report is started automatically on a node fnsfg as a cron job every Sunday at 7 PM to be ready for Monday's All Experimenters Meeting at FNAL.

2. An exhaustive report for any particular output tape can be obtained by typing

evt_per_tape OUTVSN

where OUTVSN is the output tape volume label.

3. Just to get the number of events on any particular output tape type

evt_only OUTVSN Mon

3.6 Recovering from the system crash (The Cold Start).

If the whole farm went down and was rebooted you have to:

1. Note that the first letter should be in upper case.
2. Note that all three letters should be in uppercase.

- make sure that all farm nodes are up. Ask system operators in FCC (x2746).
- Send a mail to the unix production manager unix_proman@fnsfe noting the fact and detailing the circumstances of the crash.
- Execute "after_crash" procedure by typing

  after_crash

  at **fnsfe%** prompt. This procedure would automatically restart TPM and dbl3 servers. It will also restart all network transfers and salvage all the output from the output tapes that were in the tape drives when the system crash occurred.

3.7 Blocking The Input Tape From Being Submitted on D0FS.

In order to avoid the conflicts with the user requests for the tapes on the D0FS side, a fake job is submitted on D0FS to STAGE_IN_USE_ON_UNIX_FARM queue. The jobs are remotely submitted from DZERO account on the D0FS, password.......... After the farm crash (either for the whole farm or for the individual nodes) make sure that the jobs that crashed are removed from STAGE_IN_USE_ON_UNIX_FARM queue on D0FS. Log in to D0FS, then type

show entry

Look for the job with the name YYYYYY_YYYYYY_XX_UNIX, where YYYYYY is the volume label of the tape in question. For example, the output may look like

D0FSA$ show entry

Entry	Jobname	Username	Blocks	Status
69	WM0874_WM0874_XX_UNIX	DZERO		

Executing On available batch queue STAGE_IN_USE_ON_UNIX_FARM

After finding it remove the corresponding entry:

DELETE/ENTRY=ENTRYNUMBER

where ENTRYNUMBER is the entry number, 69 in the above example.

3.8 Diagnostics - Log Files.

There are several levels of diagnostics. None of them require the operator involvement. Just in case the location of several important log files is outlined below:

3.8.1 Communications with D0FS.

$PRODIR/logdb/1992/oct/19-24/copy_results_log.OUTVSN

where OUTVSN is the output tape volume label. Here substitute 1992 for a current year, oct for the current month and 19-24 for the current date. Available date ranges are 1-6, 7-12, 13-18, 19- 24, and 25-31. Reports the status of the copying of the DSTs and RCPs to D0FS. If some problem occurs during the transmission the mail will be sent also to unix_proman mailing alias as defined in .mailrc.

$PRODIR/report/copied_dst.log

contains a list of the output tapes for which the DSTs have been copied as well as the date of the transaction.

$PRODIR/report/fmrcp_copied.log

contains a list of the output tapes for which the FATMEN RCPs were produced and shipped to D0FS as well as the date of the transaction.

3.8.2 I/O Spoolers Log Files.

Spooler log files are also in $PRODIR/logdb area. The names have the following format:

inspool_QUEID.log

outspool_QUEID.log

Log file names are a little bit different for farmlets. Since there is no more direct link between the output tape and the input tapes, the outspooler logfile is now indexed by the output tape VSN and has a time stamp, e.g.

outspool_WN2024_3020909.log

In this notation WN2024 is an output tape VSN, 3020909 is a time stamp. The timestamp shows the time when the output spooler has started and can be deciphered in the following way: the first digit stands for the year (3 for 1993), the second two digits stand for the month (02 for February), next two digits denote the date; then goes an hour (0 to 24).

3.8.3 Reconstruction Program Log Files

RECO log files can be found in $PRODIR/logdb area. The name has the following format: for the *standalone* mode:

PROJECT_QUEID.log

For example, *RECO_FULL_SGI_604.log*.

The RECO logfile for a *sequential* mode in addition to the QUEID has also the sequence number of the reconstructed file, e.g.

RECO_FULL_SGI_3074-6.log

Here we have a logfile for the reconstruction of a QUEID 3074, file number 6.

RECO logfiles for a *parallel* mode are kept per worker node in $PRODIR/wrkdb and have the following format:

wrk_WRKNODE_TIMESTAMP.log

E.g., *wrk_fnsf134_3061611.log*.

3.9 Diagnostics - Mail Messages.

In order to avoid making continuous monitoring of the system necessary most of the error conditions are being reported to an operator via mail messages sent to *unix_proman* mail alias. This alias is defined in a *.mailrc* file in the dzero *$HOME* directory. Thus new mail addresses can be added or old mail addresses can be removed by editing this file.

All mail messages can be divided into the following categories:

- **Warnings.** The TPM system has identified a problem and was able to fix it successfully by itself. Excessive repetition of a warning message may signal that there is a problem requiring attention.
- **Errors**. Need operator intervention but not necessarily immediately.
- **Alarms**. Require immediate operator intervention.

3.9.1 Warnings.

Parallel servers on $FMLNODE will be restarted A message indicating that one or more parallel servers died for *$FMLNODE* logical machine. The system recognized this and will restart the parallel shell.

Rsh got stuck in ...; killed Communications problem between the node running TPM and a remote node (remote I/O server or worker node).

RECO crashed at date *for the dataset* dataset Indicates that the reconstruction code failed on a particular dataset.

The previous intape job ended ABEND VSN N *files* If a particular tape VSN is not an urgent project - ignore. Otherwise if N = -1 or N = 0 (which indicates that no files were read from that tape) you can change the status of the tape in question from **SUBMITTED** to **WAITING** in *$PRODIR/resources/injobs*.

3.9.2 Errors

Tape VSN *is in use on D0FS* Either the tape is really in use by somebody on D0FS or it is a leftover from a system crash. Please proceed per instructions in *Section 3.7 on page 13.*

The previous outtape job ended ABEND VSN N *files* If N is less than or equal to zero - return this tape to spool of available tapes by changing **USED** to **READY** in *$PRODIR/resources/blanks*. Otherwise please transfer the DSTs from this tape to D0FS by either executing the GUI widget *copy_dst* or typing

```
cd $PRODIR/exe
pick_up_all
```

3.9.3 Alarms

All drives are busy; fatal error submitting in(out) spooler More logical machines are probably declared that there are tape drives in the system. Check the *$PRODIR/resources/resource* and *$PRODIR/resources/drives.CPU* files. If it is still not clear what happened - page D0 immediately.

No available output tapes Add more tapes per instructions in Section 3.3 on page 12.

This list is not exhaustive. Most of the remaining error messages fall into the **Warnings** category or require some explicit action.

3.9.4 Automatic Paging System

In order to facilitate a quick response from the shift personnel certain problems were considered important enough to create an automatic paging system (see Section 2.2.3 on page 3, *shift_alarm.csh* script). Below is a table of error codes displayed on the pager and their meaning:

TABLE 1. Alarm Codes

Code Number	Code Meaning
000001	Disk overflow; /dzero is above 95%
000002	No available tapedrive; fatal error submitting the spooler
000003	Error count exceeded for a tapedrive
000004	All drives are busy; same as 000002
000005	All drives are busy; fatal error submitting the output spooler
000006	All drives are busy; fatal error submitting the input spooler
000007	No available blank tapes; If you're an FCC operator - execute add_blanks as described in Section 3.3 on page 9; if you're a D0 shift personnel - check with FCC operators that they've added new blank tapes.
000008	No project definition for a given project. Check that an appropriate PROJECT file exists and named appropriately (see Section 4.1 on page 13 for details).
000009	One of the spooling areas is above 89% utilization; check your mail to find out which.

4.0 What Is A Project, How To Define And How To Switch It

A project is a group of reconstruction conditions applicable to a processing of a particular dataset. E.g., the tracking done may be a road tracking or a full tracking, databases used are different for the collider data and Monte-Carlo data, etc.

4.1 How to define a project

A project is defined in a file PROJECT.project, where project is a project's name.These files are located in $PRODIR/project area. The PROJECT.project is an ascii file containing certain tagged fields describing the reconstruction conditions. Current the tag words include the following:

PROJECT - name of the project

LIBPROD - defines the area from which reconstruction RCP-files will be linked

VRSN - version of the reconstruction code to use

PASS - pass release of the reconstruction code to use

PRC - reconctruction identifier for FM purposes (like REU for regular processing, RFU for full tracking, etc.). This is decided by FATMEN people.

SETUP - which setup file to use for linking of the RCP-files.

EXEC - name of the executable to use.

RCP - name of the file to be used as d0reco_rcp (main RCP-file controlling the reconstruction). You can also encounter separate tags **RCP_P** - for *parallel* mode, **RCP_Q** - for *sequential* mode, and **RSP_S** - for *stand-alon*e mode.

FLAGS - any special flags necessary for the reconstruction. Not used.

An example of such file follows for **EXPRESS_SGI** project:

```
PROJECT EXPRESS_SGI
LIBPROD full_d0reco
VRSN 10
PASS 13
PRC REU
PRC REU
SETUP $PRODIR/project/reco_full/setup_10_12.csh
EXEC $PRODIR/exe/full_d0reco_10_13.x
RCP $PRODAT/$FMLNODE/project/d0reco.rcp
FLAGS
```

4.2 How to switch a project

To switch a project you only need to change the project field in the *$PRODIR/resources/resource* field. The system will automatically finish with the previous project and then switch to a new one. Please note that if a tape for the previous project has just been started it may take a certain amount of time before the tape for the new project will be submitted.

5.0 How The Bookkeeping Is Done

This section describes in details scripts that are used to produce the RCP-files. RCP-files are ascii data file formatted so that D0 RCP-manipulation package can be used. In a context of TPM these files serve to enter the information about the processed sets into a FATMEN database and a production database. This RCP-files are also kept in *$PRODIR/pdbkd* in a merged form (per 6 days period).

5.1 RCP- creator *conv_prepare_rcp*

Calling sequence: conv_prepare_rcp <vsn> <pdb area> <proman linktemps>

The overall purpose of this script is to make rcp files for the processed datasets on the given vsn, and to send them successfully to D0FS.

5.1.1 Updating stat.dir and stat.rcp.dir.

A method of locking files while appending information to them is used frequently throughout this script. This is done with the source code *append.x* by setting the environmental variable TARGET to the file that is to be updated, and ARROW to the temporary file that is to be appended into the TARGET file. The first two instances of this are used to update the stat.dir in the year directory of pdbkd, and the stat.rcp.dir in the local week directory of *pdbkd.*

5.1.2 Finding out processed datasets.

The csh script *outpdb_report.csh* is used to put the names of each processed dataset on the given vsn into the file defined by the environmental variable **OUTSET**. This file is used throughout this procedure.

5.1.3 Preparing ftp files and executable rcp removal files.

The two environmentally defined files **DELSTA** and **DELDST** are created to make executable files to remove rcp and dst files after they have been copied to D0FS. The environmentally defined files for ftp to D0FS are defined as follows: **OUTDLG** - Sends dst rcp files to D0$DATA$BUFFER, **OUTSLG** - Sends sta rcp files to usr$root:[fmd0.rcpfarmin], *OUTRDB* - Sends sta and dst rcp files to usr$root:[prod_db.prod_db.sgidata] if not monte carlo files. If they are monte carlo files, they're sent to usr$root:[prod_db.prod_db.mc]

5.1.4 Making individual rcp files.

The rcp files are now made into the designated proman linktemps area. If they're monte carlo files, then the script fm_mcrcp.csh is used to make these rcp files. If they're not monte carlo, the script *fm_rcp.csh* is used. The only difference between these two scripts is in making the generic names.

5.1.5 Making merged rcp files.

The script *rcp_merge* is used to make merged rcp files. This script is called twice, once to make the merged sta rcp files, and once to make the merged dst rcp files. There can be at most 40 individual rcp files merged into one merged rcp file. If there are more than 40 datasets on the given vsn, then there will be more than one merged file. The naming convention used to distinguish multiple merged files is '..._1_...', '..._2_...', etc.

5.1.6 Transfer to D0FS.

Now D0FS is checked to be up. If not, the process will loop until it is up. When D0FS is ready, ftp is executed with the previously made ftp files. When the transfer is done, the script *check_transfer_rcp* is called three time to check the transfers. The first time it is called to check the transfer of sta rcp files. Each sta rcp file successfully transferred has it's filename entered in the executable file DELSTA to be deleted. The second time it is called to check the transfer of dst rcp files. The successfully transferred files have their filename's entered into DELDST to be deleted. The third time it is called to check the transfer of the merged rcp files. If successful, their names are entered into DELFM to be deleted.

5.1.7 Updating more info into pdbkd.

The script *checkmark.csh* is called twice, once for sta, and once for dst, to append the staf.dir and dstf.dir in the year directory in pdbkd. This is done using the TARGET and

ARROW appending procedure previously described. Then the script *update_dststa.csh* is similarly called twice to update in the week directory of pdbkd. Here, the *stat.dir, dstf.dir, fm.dir, stat.rcp, staf.rcp* and *dstf.rcp* files are appended.

5.1.8 Record files for production database

Here, a file named *RECO_SGI.<current-date>*, has added to it the name of each processed dataset on the given VSN and the number of records in it. These files get transferred over to D0FSevery day by a process independent of *conv_prepare_rcp*. The process that does this transfer is *$PRODIR/exe/copy_rcp_merge*.

5.1.9 Delete rcp files

Now the rcp file are deleted from their *$PRODAT/proman/linktemps* areas.

5.1.10 Rename EXPRESS dst files in buffer

If the datasets are EXP stream, then the dst files in the buffer in d0fs are renamed to have the prefix AAA_.

5.2 How The RECO Histograms And Summaries Are Kept

- Please note that effective December 26, 1994 production of histograms and summaries has been discontinued; They are directed to /dev/null in *wrkshell* script. However all machinery exists as described below; please uncomment the corresponding lines in *wrkshell* to resume histograms and summaries archiving.

Two csh scripts and one fortran source file are used in the process of merging. The overall goal is to merge together all the histogram or summary files of similar runs. In the case of histograms, the files to be merged are located in the spool00, spool01, and spool02 *histograms* directories (e.g., */spool00/dzero/fnsfe_0/histograms)*and the current week directory of */proman/hstdb.* In the case of summaries, the files to be merged are located in the spool00, spool01, and spool02 *summaries d*irectories (e.g., */spool02/dzero/fnsfe_2/summaries*) and the current week directory of */proman/sumdb*. This merging is done on a daily basis. The general algorithm for merging histograms and summaries is the same. For merging histograms, the two csh scripts used are *hist_merge.csh* and *hist_merge_remote.csh*. These scripts are located in *$PRODIR/exe*. The fortran source code is *h_gram_-merge.x.* For merging summaries, the two csh scripts used are *summ_merge.csh* and *summ_merge_remote.csh.* The fortran source code is *sum_merge.x*. All of these files are located in *$PRODIR/exe*.

5.2.1 hist_merge.csh - summ_merge.csh

The first script called using crontab is either *hist_merge.csh*, in the case of histograms, or *summ_merge.csh*, in the case of summaries. The first thing this script does is to check the file *$PRODIR/resources/resource* to find out which nodes are labeled **FARMLET**. If any are found, the files *resource.<node_name>* in the same directory are checked for each such node to find out the directory in which the files to be merged reside. Then, using rsh, *hist_merge_remote.csh* in the case of histograms, or *summ_merge_remote.csh*, in the case of summaries, is executed remotely on their nodes for each node found. The first argument for this call is the directory name and the second argument is the node name.

5.2.2 hist_merge_remote.csh -

summ_merge_remote.csh

The goal of these scripts is to set up an environment to execute the merging source files. In principle, both *h_gram_merge.x* and *sum_merge.x* are used the same way. They each take in two environmental variables. The first variable has the name of a file which contains a list of the filenames to be merged. The second variable is the name of the file in which the final merged output is to be put into. For histograms, the name of the first environmental variable is **HIST_LIST**, and for the second, **HIST_MERGE** is used. For summaries, the first variable is **SUM_IN**, and the second is **SUM_OUT**. The first thing the **remote.csh* (where * stands for hist or summ) script does is get each run number from the files in the directory it was given as it's first argument. It then loops over each run number. For each run number, it's WM tape label is found. This is used to find out which stream the run has. Then, each file name found with this run number is put into a file with the naming convention *<stream>_<run_no>_HIST.TEMP.<node_name>* . This filename will be defined as the first environmental variable. Then the current week directory in */proman/hstdb* area, in the case of histograms, or */proman/sumdb* area, in the case of summaries, is checked if there are any files of the current run number to be merged residing in them. If there are, their names are also put into the **HIST.TEMP** file mentioned above. If none are found, the directory of the previous week is checked in the same manner. The filename for the second environmental variable is given the naming convention *<stream>_<run_no>.HIST.<node_name>* and is in the current week directory of */proman/hstdb,* in the case of histograms, or */proman/sumdb*, in the case of summaries. The merging source file is now executed. If it's successful, then all the individual files that were merged are deleted. If not successful, the files are not deleted, and a message is appended to the file */proman/hstdb/err_hist,* in the case of histograms, or */proman/sumdb/err_sum*, in the case of summaries.

6.0 Graphics User Interface (GUI) based on SGI Explorer

6.1 Introduction.

The Graphics User Interface was developed to simplify the monitoring for a person not familiar with the TPM intrinsics. SGI Explorer system is used along with Motif and SGI GL. For more information about Explorer, please refer to Explorer User's Guide and Explorer Module Writer's Guide.

Production. map is a collection of modules performing a specific monitoring task. There are 8 modules currently, their functions are presented below:

6.2 ControlPanel Module.

It provides two option menus. One is *Farmnodes*, containing all the logical machines available. The other is *Utilities,* providing a node-independent tool displaying job status or performing a task.

Farmnodes fnstfj fnsfe_0 fnsfe_1 fnsfe_2

Utilities:

- *tpm_disp*: displays relevant information from jobs.log in 80-column format.
- *tpm_wide*: displays relevant information from *jobs.log* in 132-column format
- *tpm_rsrc*: displays resource file
- *PickupAll*: see CopyDST
- *PickupOne*: see CopyDST

For *tpm_disp* and *tpm_wide*, the colors are used to show different status of a given job.

- GREEN: *ended*
- CYAN: *inqueued*
- RED: *failed_on_insp*
- MAGENTA: *success*
- YELLOW: all the others except the above

6.3 Subcontrol Module.

This module provides a menu for each logical machine in the ControlPanel module.

6.4 Farmnodes Option Menu.

- *Tapes-->MonitorTape*: displays tape labels and drive names if the tapes are currently mounted and the inspooler and the outspooler are running on the logical machine.
- *Resources-->MonitorResource:* it displays the *mode* (**PARALLEL**, **SEQUENTIAL**, or **STANDALONE**), *status* (**FARMLET**, **UNAVAILABLE**, or **READY**) and *project* for the logical machine. Also it can change the any of these fields. Just type in the text slot you want to change. Remember to type from LEFTMOST of the slot and type <RETURN> when you have finished.
- *WorkerNodes-->MonitorWrknodes*: it checks if d0reco.x is running on worker nodes attached to the logical machine. Three colors are used to display different status of the worker node.

  GREEN: running (CPU time for d0reco.x increases within 5 seconds)

  RED: idle (no d0reco.x is running)

  YELLOW: hung_5s (CPU time for d0reco.x does not change within 5 seconds)
- *STAsize-->MonitorSTAsize*: it has two options (**Timer On** and **Timer Off**) and two text slots (**Delay** and **Current Value**). When **Timer On** is selected, it will be triggered by a Timer module. The Timer module contains Start Value, End Value, Step, Delay and Current Value (this module is part of the Explorer library distributed with Explorer). When Timer Off is selected, the module will fire only if STAsize in Subcontrol is selected. You can change the Delay value as you like. The module checks the STA file size being processed every 3 seconds until the 30% of the Delay value is reached. If the Delay value is 30 seconds the Module will sample the STA file size every 3 seconds, totaling 0.3*30/3 times. All these values can be changed easily by typing in the corresponding text slots or changing the dials.

6.5 Utilities Module.

Provides a list of options in ControlPanel Utilities menu (refer to ControlPanel section). It also contains two text slots to display related information about the farm nodes. The function of each text slot is outlined below:

- *Physical Nodes*: a physical node name
- *Processes*: a UNIX shell command

6.6 CopyDST Module.

- *PickupOne*: It copies failed DSTs to a specific output tape. Just type in label of the tape to which DSTs were copied in failure.
- *PickupAll*: It copies all failed DSTs on all machines.

6.7 How To Set Up And Run Production Map.

- Add or modify file *.explorerrc* in your home directory. For an example look for it in *~denisenk* on D0SGI.
- Add the following to your *.login* file:

setenv D0FARMHOME /d0sgi0/data0/d0farm_monitor
set path=(\$path /d0sgi0/data0/d0farm_monitor)

- Type

  explorer

  Note: if it doesn't work define

  alias explorer /usr/explorer/bin/explorer

  or do

  setup minexp
- When Explorer menus appears, click on *Categories* in localhost window and choose *Production,* and a new menu titled *Production* will appear. Press left mouse button, dragging production (highlighted) in the menu into the Map Editor window, and drop it. Production map will automatically start executing (*firing* in Explorer slang).

In addition, the control panels of all modules have been carefully arranged in such a manner that all of them can be spotted easily while minimizing overlap between them. Just place the Map Editor in the lower left corner of the screen and drop production.map in the upper left corner of the Map Editor. Of course, you can always rearrange them as you wish.

Important: When you type in a text slot, be sure to press <return> after you have finished typing. When a module is firing, its windows will not resize and the module in the Map Editor will be yellow on a color monitor.

A special account (username *operator*, password ....) has been set up for the purpose of using this GUI. The executables and source codes are in /d0sgi0/data0/d0farm_monitor directory on D0SGI. The source codes are in C programming language.

7.0 TPM Production Manager Scripts List

This description starts with the top level script and describes it in a tree level fashion.

7.1 tpm_submit_job

(in *$PRODIR/tpm*). Top level scripts; handles the job submission to all the nodes; maintains process table; submits auxiliary scripts (like *start_filter_job*).

7.2 start_filter_job

(in *$PRODIR/tpm*). Checks that the tape with filtered files is mounted on fnsfe; in case the tape is finished starts a script that creates RCP-files for the filtered datasets on tape and copied them to D0FS.

7.2.1 copy_results_fl.csh

(in *$PRODIR/exe*). Handles creation of the RCP-files for the filtered datasets; copied the RCP-files to D0FS.

7.2.2 conv_prepare_rcp

(in *$PRODIR/report*). Gets the datasets from the finished filtered tape STAT Rcp-file; calls *fm_rcp.csh* to create a FM-RCP file for each dataset.

7.2.3 fm_rcp.csh

(in *$PRODIR/report*). Actually creates FM-RCP files; using FLTS RCP-files to extract information.

7.3 update_resource_table

(in *$PRODIR/tpm*). Updates the resource table for stand--alone nodes; checks that either reconstruction is running or the node is ready for the next job; cleans up after the job has finished or crashed; finally changes the status of the node in the resource table (BUSY, READY, DOWN, or SPOOLER).

7.3.1 start_job_now

(in *$PRODIR/tpm*). Submits a tape job to the stand-alone node; in this case tape mounting is handled by the stand--alone node itself.

7.4 update_farmlet_table

(in *$PRODIR/tpm*). Handles the job submission for farmlets (I/O server nodes with worker nodes attached). Makes disposition of available tape drives. Submits shells to the I/O servers; mounts input and output tapes on the I/O servers; copies and catalogues the results of the finished jobs. Handles interaction with FM via stage_in_use_on_unix_-farm queue on D0FS.

7.4.1 start_mu_shell

(in *$PRODIR/tpm*). Starts a remote shell on a I/O server node of the farmlet; the shell is configured in correspondence with a project requested.

7.5 copy_results.csh

(in *$PRODIR/tpm*). Copies the DSTs for the finished tape; creates and copies to D0FS RCP-files for STAs and DSTs.

7.5.1 copy_dst

(in *$PRODIR/report)*. Copies the DSTs to D0FS according to the STAT RCP-file for the tape in question.

7.5.2 uptest

(executable image in *$PRODIR/exe*; source available in *$PRODIR/sources*). Checks that D0FS is up and running; also checks that FM and PROD_DB areas are writable.

7.5.3 check_buffer.csh

(in *$PRODIR/exe*). Checks that D0$DATA$BUFFER area on D0FS has an adequate space to accept DSTs. Currently set to 600 Mb.

7.5.4 check_transfer.csh

(in *$PRODIR/report*). Checks the status of the DST transfer via zftp.

7.5.5 conv_prepare_rcp

(in *$PRODIR/report*). Creates and copies STA and DST RCP-files.

7.5.6 fm_rcp.csh

(in *$PRODIR/report*). Creates FM RCP-file for a given DSN.

7.5.7 fm_mcrcp.csh

(in *$PRODIR/report*). Creates FM RCP-file for a given DSN that conforms to D0 Mote-Carlo filename conventions.

7.5.8 check_transfer_rcp

(in *$PRODIR/report*). Checks the RCP-files transfer status.

7.6 start_farmlet_job

(in *$PRODIR/tpm)*. Mounts an input or output tape on a remote farmlet I/O server. Marks appropriate tape VSNs as used; updates process table.

7.6.1 check_invol.csh

(in *$PRODIR/exe*). Checks whether an input tape is already in use on D0FS; if yes returns -1. If no then submits a pseudo job to stage_in_use_on_unix_farm queue on D0FS to let FM know that the tape is in use on the farm. In the latter case it returns queue entry number.

8.0 TPM For Streaming

Being the general tool for production on Unix machines TPM can be easily adapted for streaming. The streaming operation includes reading previously reconstructed datasets (STA) and writing selected by the filter program events to different output streams. Thus this operation is I/O rather than CPU intensive since it takes comparable time to spool and process a dataset. With respect to TPM operations are very similar to operations in a sequential mode on the farms. The following paragraphs cover subtle differences between the streaming and farm production.

8.1 TPM Scripts For Streaming.

The top-level script *tpm_submit_job* is identical to the farms. The rest of the scripts have their streaming equivalents: *update_streamer_table* corresponds to *update_farmlet_table*, *start_streamer_job* corresponds to *start_farmlet_job*, etc. The biggest difference conceptually between the farms production and streaming is that a new logical machine is introduced: so called “S” LM that is responsible for the output on the streaming stations. Thus the input for the streaming is handled by the regular LMs whereas the output, which in the case of streaming includes many output tapes, is handled by the “S” machine. This “S” machine also supports the running of the *omni_mnf.csh* (which is the functional equivalent of the *d0reco_mnf.csh*) and the running of the *merge*-utility described below.

The *resource* file for the streaming looks very close to the *resource* file for the farming, with project being set to **STREAMING**, and status to **STREAMER** (as opposed to **FARMLET** for the farming).

8.2 Merge Utility.

In order to reduce the number of the output files we are dealing with (each input file spawns N output files, where N is the number of streams) the files are merged per run. There are several criteria used to decide which files and when to merge. Firstly, there is a minimum number of files necessary to initiate merging. Secondly, there is a maximum number of hours which any individual file can stay on disk without being merged. Thirdly, if the space left on the output disk is smaller than the predefined number of blocks, the merging will be done for all files currently on disk until the disk utilization goes down. All these parameters together with the stream names are defined in the file *stream_names.$FMLPHYS* residing in *$PRODIR/resources* directory. This file looks like the following:

```
WZS01 X_STA01ZZZZZZZ_ALL00_WZSXX02 1 25 500000 8:

TOP01 X_STA01ZZZZZZZ_ALL00_TOPXX04 2 25 500000 8

FAK01 X_STA01ZZZZZZZ_ALL00_FAKXX02 2 25 500000 8

QC101 X_STA01ZZZZZZZ_QC102_NONEX00 3 25 500000 8

MIN01 X_STA01ZZZZZZZ_MIN02_NONEX00 3 25 500000 8

MU101 X_STA01ZZZZZZZ_MU102_NONEX00 4 20 500000 8

RGE01 X_STA01ZZZZZZZ_ALL00_RGEXX05 5 20 500000 8
```

Here the first field is the name of the stream, the second field is the extension of the streamed STA for this file, the third field is the number of the outspooler, the fourth field is the minimum number of files to be merged, the fifth field is the minimum free space on the output device before the merging starts, and the last field is the maximum time a file can stay on disk before being merged (in hours).

The third parameter (a number of the output spooler) allows to combine several output streams on one output tape. The number also points to a file *outspool_list_#.$FMLPHYS*, that contains the actual device directory name from which the outspooler will copy files to tape.

8.3 Streaming Data Flow.

The streaming executable (or *omni-filter*) is running on the same physical node on which the I/O is performed. Each logical machine has its own home area on a separate disk. The location of this area is as usual pointed to by *$PRODAT* area which can be obtained from the file *resource.$FMLNODE* residing in the *$PRODIR/resources*. In this area there are two subdirectories, *inspool* and *raw*. An input spooler (which number is defined in *resource* file) spools input files into the *inspool* area. The next file to be processed is moved from *inspool* to *raw*. The resulting streamed STAs are put into the subdirectory of the *$PRODAT* area of the “S” machine (there can only be one “S” machine per physical node). The name of this directory is *stain*. The rcp-files created by the *omni_mnf.csh* are stored in the same directory in the files with *.done* extension. This area serves as an input for the *merge*-utility. After merging occurred the resulting merged STAs are placed into the directories defined in the *outspool_list_#.$FMLPHYS* for the particular stream. The merged RCP-files are kept in the *$PRODAT/$FMLNODE/rcpout* directory. These RCP-files are used to create FM RCP-files when the tape is finished. A script responsible for this is *$PRODIR/exe/copy_stream_results.csh*, which has the same arguments as *copy_results.csh* used on the farms.

9.0 Acknowledgments.

I’m grateful to Mark Galli who contributed to the Bookkeeping section and Hailin Li (who wrote the GUI).

Advice and support of Lee Lueking and Adam Para is much appreciated. Paul Lebrun helped us with learning Explorer.